Computer Vision Based Navigation for Spacecraft Proximity Operations

by

Brent Edward Tweddle

B.A.Sc., University of Waterloo (2007)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2010

Author .....
Department of Aeronautics and Astronautics
January 29, 2010

Certified by .....
David W. Miller
Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
Alvar Saenz-Otero
Research Scientist, Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Eytan H. Modiano
Associate Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Computer Vision Based Navigation for Spacecraft Proximity Operations

by

Brent Edward Tweddle

Submitted to the Department of Aeronautics and Astronautics
on January 29, 2010, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

The use of computer vision for spacecraft relative navigation and proximity operations within an unknown environment is an enabling technology for a number of future commercial and scientific space missions. This thesis presents three first steps towards a larger research initiative to develop and mature these technologies.

The first step that is presented is the design and development of a "flight-traceable" upgrade to the Synchronize Position Hold Engage Reorient Experimental Satellites, known as the SPHERES Goggles. This upgrade enables experimental research and maturation of computer vision based navigation technologies on the SPHERES satellites.

The second step that is presented is the development of an algorithm for vision based relative spacecraft navigation that uses a fiducial marker with the minimum number of known point correspondences. An experimental evaluation of this algorithm is presented that determines an upper bound on the accuracy and precision of this system.

The third step towards vision based relative navigation in an unknown environment is a preliminary investigation into the computational issues associated with high performance embedded computing. The computational characteristics of vision based relative navigation algorithms are discussed along with the requirements that they impose on computational hardware. A trade study is performed which compares a number of different commercially available hardware architectures to determine which would provide the best computational performance per unit of electrical power.

Thesis Supervisor: David W. Miller
Title: Professor of Aeronautics and Astronautics

Thesis Supervisor: Alvar Saenz-Otero
Title: Research Scientist, Aeronautics and Astronautics

Acknowledgments

First of all, I would like to thank my advisors Professor David W. Miller and Dr. Alvar Saenz-Otero for their guidance, advice and support. Additionally, I would like to thank the National Sciences and Engineering Research Council of Canada's Post Graduate Scholarship Program as well as the Gordon M. MacNabb Scholarship Foundation for their generous financial support.

I would also like to thank Professor John J. Leonard and Professor Berthold K.P. Horn for their advice on critical sections of this thesis. Additionally, I would like to thank Dr. Glen Henshaw and Stephen Roderick of the Naval Research Laboratory for their collaboration and support in the LIIVe program. Joe Parrish, Javier De Luis, Joanne Vining and John Merk from Aurora Flight Sciences have been extremely helpful in this project.

The entire SPHERES team at MIT has been immensely helpful in preparing this work. Specifically I would like to thank my office-mate for two years, Jacob Katz, for all of his help. Additionally, Paul Bauer was in incredible help for a lot of the work in this thesis.

Thanks to Swati Mohan, Christophe Mandy, Simon Nolet, Jaime Ramirez, Amir Fejzic, Jack Field, Enrico Stoll and Chris Pong who have all helped and taught me so much about SPHERES. A lot of the work in this thesis would have never been possible if it weren't for two great UROPs Simon Calcutt and Edward "Chip" Whittemore. I definitely appreciate the help Joe Robinson, John Richmond, Zack Bailey, Matt Smith, Alessandra Babuscia and everyone else in the Space Systems Laboratory. Marilyn Good and Sharonleah Brown have been a great help for getting things done around the lab.

I am also grateful the people who supported me at Waterloo. The support of Professor David Wang and Professor Carol Hulls is what brought me to MIT in the first place. I would like to thank all of the members of the University of Waterloo Aerial Robotics Group over the years, especially Steve Buchanan, Matt Black, Jason Gillham, Jake Lifshits, Sebastian Peleato, Jason Dyck and Tristan Lall.

Lastly, I would like to thanks to my family and friends. My parents and grandmother, who send their love and support from so far away. To my new friends here in Boston and my old friends back in Canada, thanks for everything. And last but not least, to Lindsay Hays, the best girlfriend a guy could ever have: here's to year three!!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

From the first spacecraft docking maneuver that was performed on Gemini 8, until the recent technology demonstration of DARPA's Orbital Express, spacecraft proximity operations have been an integral component of a variety of space mission objectives. In order to achieve these objectives, any mission that involves proximity operations must have an accurate, reliable and robust relative guidance, navigation and control (GN&C) system. Numerous types of commercial and scientific space missions require the ability to operate in close proximity to another object, without colliding with it. A few examples of these are satellite servicing and repair, on-orbit assembly, asteroid sample and return, precision planetary landing and surface rovers, just to name a few.

1.1.1 Previous On-Orbit Navigation Techniques for Proximity Operations

Over the years a variety of sensor technologies have been used for proximity navigation. One of the earliest on-orbit autonomous navigation systems was the Russian Kurs system, which is still used for rendezvous navigation by the Soyuz and Progress vehicle. The Kurs sensor system is based on an S-band radio transponder that mea-

sures range, range-rate, relative pitch and relative yaw [24].

More recently, ESA's Automated Transfer Vehicle (ATV) has demonstrated the use of a Relative Global Positioning System (RGPS) for the 30 kilometer to 500 meter phase of docking with the International Space Station (ISS). The RGPS system uses raw measurements from two GPS receivers (one on each vehicle), and differences these measurements in an onboard navigation filter to produce relative range and range-rate measurements[24].

The Laser Camera System (LCS) was developed by the Canadian Space Agency (CSA) to produce three dimensional maps of the Space Shuttle. It was first tested on STS-105 and has been used in all missions following the Columbia Accident, beginning with STS-114 [22]. In order to compute the three dimensional map, the LCS uses a camera to photograph the pattern created by a scanning laser that is moved along an unknown surface.

On DARPA's Orbital Express mission, the Advanced Video Guidance Sensor was used for docking between two spacecraft. A laser diode was used to illuminate a retro-reflective visual target that was processed by machine vision algorithms to determine the relative position and orientation between the two spacecraft[47]. An on-orbit photo of the target spacecraft is shown in Figure 1-1.

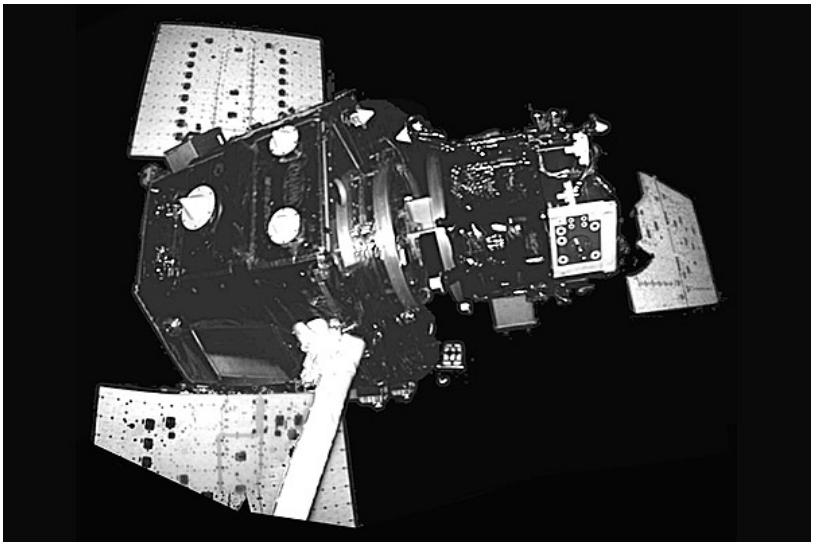


Figure 1-1: DARPA's Orbital Express On-Orbit [20]

1.1.2 Computer Vision for Proximity Operations

The use of computer vision for proximity navigation provides a number of advantages over other sensing technologies. Visual navigation sensors can be made to be small and low power, they have no moving parts and are passive systems (i.e. they require no electrical power or data from the target spacecraft). Although visual navigation has been used on a number of space missions in the past, there are many more computer vision technologies and algorithms that could improve mission capabilities, but are not yet mature enough to be implemented on-orbit. For example, in 2005 the Committee on the Assessment of Options for Extending the Life of the Hubble Space Telescope found that "...camera-based control of the grapple arm are mission-critical technologies that have not been flight-tested" and "Technologies needed for autonomous manipulation, disassembly, and assembly, and for control of manipulators based on vision and force feedback, have not been demonstrated in space"[56].

A number research projects are currently investigating computer vision based navigation for spacecraft proximity operations. The Naval Research Laboratory is developing the SUMO/FREND technology demonstration that will perform on-orbit servicing of satellites with no fiducials or grapple mechanisms[76]. Additionally, the Naval Research Laboratory is developing a Low Impact Inspection Vehicle (LIIVe), that is a small nanosatellite that is attached to a much larger spacecraft. Should a failure be detected on the larger spacecraft, the LIIVe nanosatellite would be deployed and would perform an inspection maneuver using computer vision, combined with an inertial measurement unit, as the relative navigation sensor[42].

One of the most recent missions that illustrates the state of the art in spacecraft relative navigation is the Hayabusa mission by the Japanese Aerospace Exploration Agency (JAXA), which is performing a sample return mission to the 25143 Itokawa asteroid [50] (shown in Figure 1-2). Asteroid sample return missions present a very difficult relative navigation problem. This is due to the fact that the geometry and visual appearance of the target is unknown, and the objects are usually far enough away that any human control will have a significant time delay. The asteroid landing

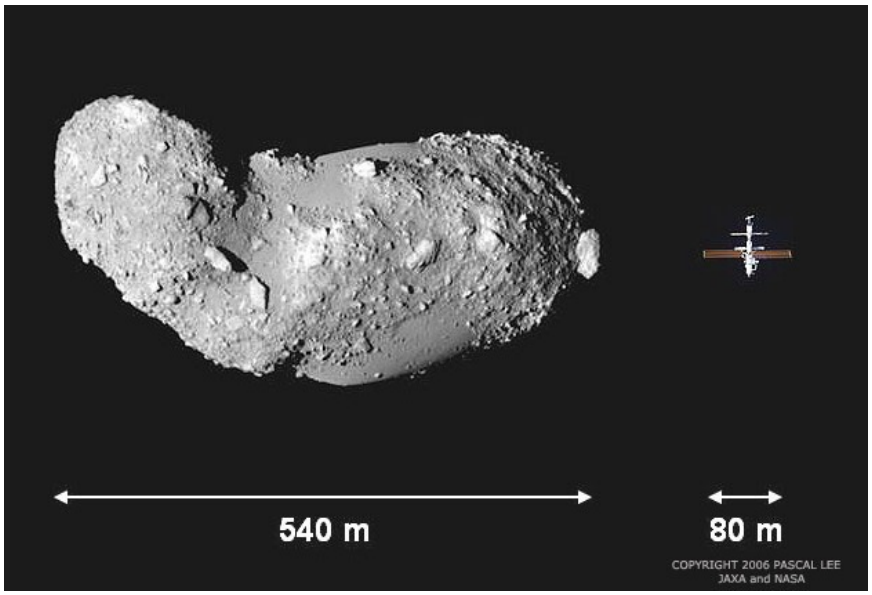


Figure 1-2: Itokawa Asteroid Relative to International Space Station [58]

vehicle, the MUSES-C, was designed to drop a visual fiducial marker onto the surface of the asteroid[100]. It had planned to use this fiducial marker to regulate the spacecraft's velocity in the horizontal plane, while a set of laser altimeters would be used to determine the altitude of the spacecraft during the touchdown maneuver. However, due to laser altimeter sensor failures, this approach was abandoned during operations and a vision based navigation algorithm, which tracked a number of point features on the asteroid, was selected and developed while the spacecraft was waiting nearby the asteroid. Since this approach was too computationally expensive to be performed onboard the spacecraft, data was transmitted to earth, and flight commands were returned with a 30 minute delay. In November 2005, the Hayabusa spacecraft made multiple touchdowns on the asteroid, however the navigation system lead to a number of errors that resulted imperfect touchdowns [101, 54, 100]. Despite this, it is likely that the spacecraft was able to collect a sample and is expected to return to Earth in 2010.

Although the Hayabusa spacecraft has successfully completed the proximity operations objectives of an extremely challenging mission, the difficulties encountered during touchdown and landing illustrate an existing unmet need for reliable, accurate and real-time on-board computer vision based navigation within unknown environ-

ments.

1.2 Outline of Thesis

One of the biggest challenges to the current state of the art in proximity navigation systems is the ability to operate within an unknown environment. The work presented in this thesis is part of an overall research initiative to develop computer vision based navigation algorithms for an unknown environment. This thesis presents the first three steps in this program: the development of a computer vision research testbed, the validation of this research testbed through the implementation of a fiducial marker tracking algorithms, and the preliminary analysis of an approach to solve the computational difficulties encountered in onboard real-time processing for vision based navigation in an unknown environment.

Chapter 2 presents the design and development of a testbed for computer vision based navigation. This testbed upgrades the Synchronize Position Hold Engage Reorient Experimental Satellites (SPHERES) with new hardware and software such as cameras, lights, high speed wireless communications and extra processing power. This hardware is referred to as the SPHERES Goggles (see Figure 1-3), and was developed in a partnership with the Naval Research Laboratory's Space Robotics Division as part of the LIIVe program.

Chapter 3 discusses the algorithmic implementation of a relative spacecraft navigation system that uses a fiducial marker of known geometry to determine the relative position, orientation, linear and angular velocities between two spacecraft. Experimental results using the SPHERES Goggles are presented, and an upper bound for the precision and accuracy of the sensor is experimentally determined.

Chapter 4 presents a preliminary study of the problems associated with high performance embedded computing for vision based relative navigation in an unknown environment. It is proposed that there is a mismatch between the algorithms typically used to implement vision based relative navigation and the computational hardware on which these algorithms are implemented. A preliminary analysis of the character-

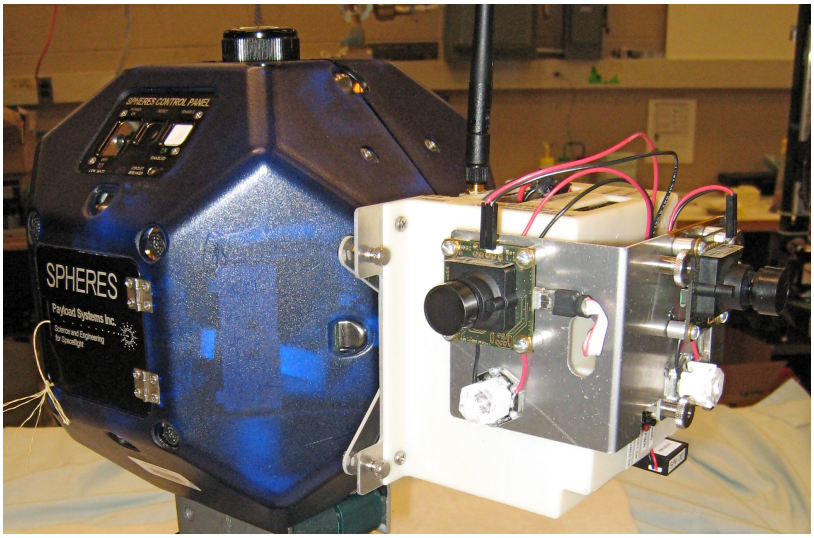


Figure 1-3: SPHERES Goggles

istics of these types of algorithms and the associated hardware that is best suited to run the algorithms has concluded that Graphics Processing Units (GPUs) are likely to provide the best computational performance for vision based navigation per unit of electrical power consumed. As a result, a potential architecture is proposed that would use stereo vision to implement dense 3D occupancy grid based simultaneous localization, mapping, and path planning onboard a GPU.

Chapter 5 summaries the conclusions and contributions of this thesis and discusses future work.

1.3 Literature Review

The literature review for this thesis is divided into three subsections (1.3.1, 1.3.2 and 1.3.3) which correspond to the topics of Chapters 2, 3 and 4 respectively.

1.3.1 Testbeds for Computer Vision Based Navigation

A number of demonstrators and testbeds have been developed to mature inspection satellite technologies. As was previously mentioned, the most recent high-profile demonstrator is DARPA's Orbital Express. In 2007, this satellite demonstrated

robotic satellite servicing with a cooperative target that utilized interfaces designed for servicing. Previously, in 2005, the Demonstration of Autonomous Rendezvous and Docking (DART) mission, attempted similar objectives as Orbital Express, however an anomaly in the navigation system caused the two spacecraft to collide. The Naval Research Laboratory is currently developing SUMO/FREND, a robotic servicing demonstrator that will use vision based navigation[76]. Another flight demonstrator is the Air Force Research Laboratory's XSS-11, which demonstrated proximity circumnavigation on orbit in 2005[103]. The AERCam Sprint was an inspector satellite that was tested on STS-87 [97]. This manually controlled inspector provided video data to an astronaut onboard the Shuttle. A follow-on program has developed the Mini-AERCam testbed, which is designed to be capable of both manual and autonomous inspection[27], however it has not yet been flown.

A number of other ground testbeds have been built to develop proximity operations technologies. For example, the SCAMP testbed was developed at the University of Maryland for semi-autonomous operation in a neutral buoyancy tank[70]. Also, the AUDASS Testbed was developed by the Naval Postgraduate Laboratory and demonstrated vision based navigation techniques on the ground[81][80]. In 2001, the Lawerence Livermore demonstrated formation flight based on stereo vision on their ground testbed[57].

The MIT Space Systems Laboratory (MIT SSL) has developed the Synchronized Position Hold Engage Reorient Experimental Satellites (SPHERES), which have been operating on the inside of the International Space Station since 2006 where they have tested a multiple guidance, navigation and controls algorithms for spacecraft formation flight [72]. Figure 1-4 shows a photo of three SPHERES performing a formation flight maneuver on the ISS.

Currently there is no testbed that can develop vision based navigation algorithms on both the ground, and in a six degree-of-freedom microgravity environment. The MIT SSL, the Naval Research Laboratory (NRL) and Aurora Flight Sciences (AFS) partnered to develop such a vision based navigation upgrade to the SPHERES satellites that is "flight-traceable" (i.e. minimal modifications will be required to qualify

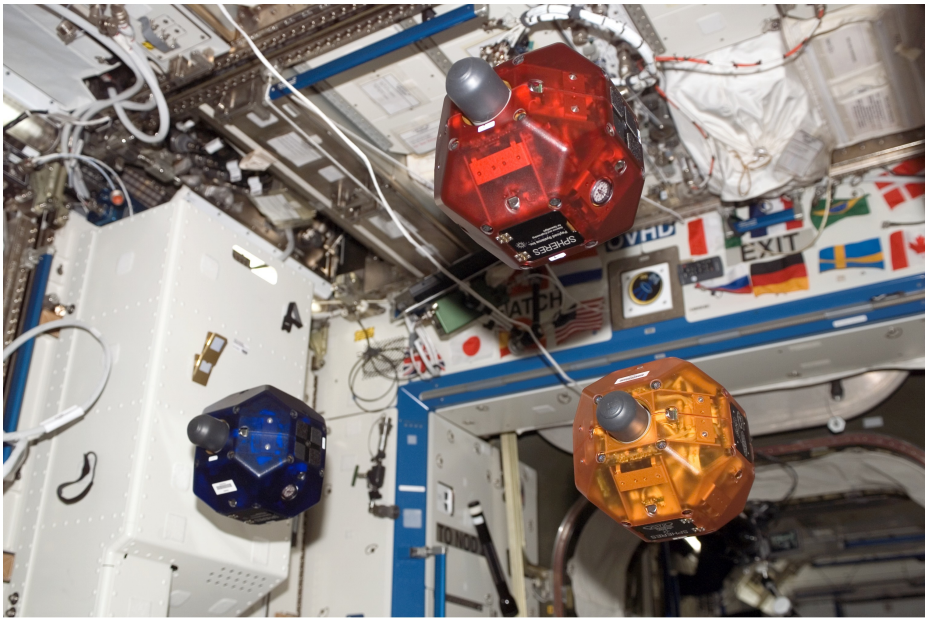


Figure 1-4: Three SPHERES in formation on the ISS [1]

the system for microgravity operations), which is discussed in Chapter 2.

1.3.2 Vision Based Spacecraft Relative Navigation using a Fiducial Marker

The problem of relative navigation using computer vision has its roots in the field of photogrammetry. Photogrammetry is the study of geometric relationships that are formed between the real, three dimensional world and its two dimensional perspective projection onto an image. It has been an area of active research since the 1800's, and its roots can be traced as far back as Leonardo da Vinci in the 1400's [14]. A modern perspective on the subject was written in 2001 by Mikhail, Bethel and McGlone [71]. Traditionally, photogrammetry has been primarily used in civil and aerial surveying, but with the advent of digital computers and low cost digital cameras, these ideas have been reapplied for computer vision applications. Current information on the field of photogrammetry is published in international journals such as *The Photogrammetric Record* and through the *International Society for Photogrammetry and Remote Sensing*. Recent texts on the application to computer vision have also been published[37, 40, 61].

There are four fundamental problems in the field of analytical photogrammetry: interior orientation, exterior orientation, absolute orientation and relative orientation [45, 37]. These problems solve for the minimum mean squared (reprojection) error estimates of certain parameters of interest using the knowledge of where a point in the three-dimensional world was projected onto the image plane. These correspondences can be either two dimensional or three dimensional points in either the image frame or the "global" real-world frame.

Interior orientation refers to the determination of the interior parameters of the camera, such as the focal length and the location of the principle point on the image plane, using a number of point correspondences. Exterior orientation (also known as resection) refers to the estimation of the rotation and translation between the camera frame and the global frame using two dimensional points in the image and three dimensional points in the global frame. Absolute orientation solves the same problem as exterior orientation, however it requires three dimensional points in the image frame (typically determined from stereo image pairs or lidar scans). Relative orientation estimates the rotation and translation between to camera orientations using two dimensional point correspondences in both image planes. In Chapter 3, the solutions to all of the problems except relative orientation will be utilized.

An currently active topic of publication is the field of augmented reality. In this application of photogrammetry, a known marker is photographed by a video camera and three dimensional computer graphics are overlaid on the video using the marker (which is free to move around in the scene) as a reference frame. The *ARToolKit* is a popular open-source library for augmented reality that is written by Hirokazu Kato [49]. This library effectively solves the exterior orientation problem in real-time.

The use of photogrammetry alone for relative vehicle navigation has a few drawbacks. Photogrammetry can only provide position and orientation and not relative velocities or accelerations. In order to do this, recursive estimation techniques are typically used [29, 18, 11]. A number of visual tracking applications have combined the fields of photogrammetry and recursive estimation, such as Kalman or Particle Filtering[4].

The application of photogrammetric techniques to spacecraft relative navigation began with the implementation of the Space Vision System by the Canadian Space Agency [33, 62]. Another photogrammetric system used in space is the Advanced Video Guidance Sensor, which was successfully used on the DARPA Orbital Express mission in 2007[47]. Additionally, the ULTOR ®Relative Navigation System was originally developed for the Hubble Space Telescope (HST) Robotic Servicing Mission by NASA Goddard Space Flight Center and Advanced Optical Systems, Inc, and was tested on the manned HST Servicing Mission 4 (SM4) in 2009.

Academic research on photogrammetric methods for spacecraft relative navigation falls into two categories, angles-only-navigation, and close proximity POSE estimation. Methods of angles-only navigation been published by David K. Geller [98] and typically falls into a class of long range visual navigation methods. Angles only navigation methods typically linearize the line-of-sight equations (see Section 3.3.1) in the state update of an Extended Kalman Filter. Another academic research program on spacecraft relative navigation using photogrammetry is the VISNAV program that was developed by Junkins [52, 35]. Junkins later developed an optimal line of sight estimation system and showed that its variance converged to the Cramer-Rao lower bound[17].

Chapter 3 will implement a photogrammetric spacecraft relative navigation system that uses a solution to the exterior orientation problem and an Extended Kalman Filter to estimate relative states using a fiducial target of known geometry.

1.3.3 High Performance Embedded Computing for Vision Based Navigation in an Unknown Environment

Computer vision researchers have developed numerous parallel methods that are designed for embedded systems [53, 79, 66]. However, given that the field of computer vision encompasses a broad range of problems, only a small subset of these methods has been developed specifically for vision based navigation applications (others focus on facial recognition, object detection etc). Researchers at the University of

Tokyo developed a good demonstration of high performance embedded computer vision based navigation. They have shown that very high performance visual control can be performed if the computer vision algorithms are accelerated by custom designed hardware processors. Their robotic arm has demonstrated the ability to throw a small object such as a cell phone in the air and catch it before it hits the ground [83]. Additionally, recent developments have shown that Graphics Processing Units (GPUs) have obtained very good performance for a number of computer vision algorithms[13, 90, 66, 30].

Preliminary work has been undertaken on hardware acceleration of computer vision algorithms that is specific to space missions. Researchers at NASA's Jet Propulsion Laboratory have implemented stereo vision algorithms on a FPGA for planetary rover applications[94, 69]. Additionally, the Autonomous Lunar Hazard Avoidance Technology (ALHAT) program at NASA has begun work on implementing terrain relative navigation algorithms on massively parallel computers [93].

The problem of computer vision based navigation and mapping in an unknown environment has been extensively studies in terms of Structure From Motion [48], Visual Odometry [77] and Visual Simultaneous Localization and Mapping [16, 46]. Structure from motion is a bundle adjustment technique that computes the motion and structure of a camera using non-linear minimization algorithm that minimizes the mean-squared distance between points across a number of images. The problem with structure from motion is that this procedure is not recursive and as a result is not commonly used as a primary method of solving real time navigation problems. Visual Odometry is a method of navigation through the tracking of feature correspondences between frames[68]. New features are added as they come into view and old features are deleted when they are no longer visible. This method has been extensively used on the Mars Exploration Rovers (MER) and is planned for the Mars Science Laboratory (MSL) [63]. The fundamental drawback of this method is that it will tend to drift over time and is unable to correct itself if an old location is revisited. Visual Simultaneous Localization and Mapping (SLAM) is the most general solution to the problem of navigation and mapping as it stores the features that were seen at past locations so

that if that feature is seen again, the entire trajectory can be corrected in a loop closure routine. The primary downside of SLAM algorithms is that its computational performance decreases as the number of features increases. The following subsection reviews the key technologies associated with a Vision Based SLAM system and its historical development.

Historical Development of Vision Based Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) was first proposed by Smith, Self and Cheeseman in their 1990 seminal paper [86]. This paper presented a stochastic framework in which robots could estimate spacial relationships between objects in their environment, as well as the robot's own location. One of the first implementations of SLAM was published by Leonard and Durrant-Whyte 1992 [60], where the authors demonstrated the ability to build a map of a robot's environment in two dimensions using only a time of flight sonar measurements. This approach implemented the stochastic framework proposed by Smith, Self and Cheeseman in an Extended Kalman Filter (EKF).

One of the fundamental limitations of the EKF SLAM is its assumption that the posterior distribution is Gaussian. It is easy to show that if the process model is non-linear, there is no guarantee that the posterior distribution will also be Gaussian (even if the prior distribution is Gaussian)[87]. In 2003, Thrun proposed the use of a Rao-Blackwellized particle filter representation for problems that have non-Gaussian distributions (i.e. non-linear process models), and formulated an algorithm known as FastSLAM[73].

Another fundamental limitation of the original Smith, Self and Cheeseman approach is that the environment is represented as a set of "objects" or "features" and their respective locations. This creates a sparse map that can be difficult to incorporate in planning and control methods since there are inherent "information gaps" between the objects. In order to avoid this problem, an occupancy grid map is commonly used, which divides the environment up into a grid and stores a probability of

occupancy at each grid location. This approach stores much more information about the environment. However this method can become much more computationally complex as the size of the occupancy grid grows; for example, a square three dimensional occupancy grid, with 1000 elements on each side, will require 1 GB of RAM. It is fairly common for two dimensional occupancy grid maps to be created from scanning LIDAR sensors mounted on a mobile robot platform[87].

A key problem in the implementation of any occupancy grid map is how to correlate two scans taken from different locations. This is commonly known as scan registration, which is commonly solved using the Iterative Closest Point algorithm that was originally proposed by Besl and McKay[7]. Recent work has been performed to develop methods for creating three dimensional grid maps based on laser range finders [23], however it is noted that the run time for these algorithms ranges from 10 seconds to two minutes per estimation cycle.

The use of computer vision for both localization and mapping (SLAM) has been pioneered by Davison [21], who fully developed inverse depth parameterizations of features for single camera SLAM systems [16]. Also, recent developments have created real-time implementation of a stereo camera system for six degree-of-freedom SLAM, using a standard laptop computer[78].

Chapter 2

Design of a Testbed for Computer Vision-Based Navigation

2.1 Overview

This chapter presents the design of the SPHERES Goggles, a hardware upgrade to the SPHERES satellites that adds cameras, lights, additional processing power and a high speed wireless communications system. The design approach is presented from a systems engineering perspective, which begins with a high level systems architecture trade study, followed by a definition of subsystem requirements, and the selection of individual components. The design of the electronics, optics and software subsystems is described individually. The fabrication of a "Flat Goggles" prototype and an packaged "Integrated Goggles" system is also presented.

2.2 Goggles System Trade Study

2.2.1 Primary Objective

The main objective of the SPHERES Goggles is to **provide a flight-traceable platform for the development, testing and maturation of computer vision-based navigation algorithms for spacecraft proximity operations**. It is im-

portant to note that although this hardware was not intended to be launched to orbit, it was required to be easily extensible to versions that can operate both inside, and ultimately outside the ISS or any other spacecraft.

2.2.2 SPHERES Constraints

Due to the fact that the Goggles must be mounted on the SPHERES expansion port a number of constraints were introduced.

1. The Goggles **must** not alter the SPHERES center of mass by more than 2 cm. Therefore the Goggles **must** weigh less than 1 kg.
2. The Goggles **must** not block the SPHERES CO2 thrusters.
3. The Goggles **should** not block the SPHERES ultrasonic receivers.
4. The Goggles **must** not consume more power than the SPHERES expansion port can provide **OR** the Goggles **must** provide its own battery power.

2.2.3 Functional Requirements

Using the primary objective of the SPHERES Goggles, a number of high-level functional requirements were derived.

1. The Goggles **must** be able to image objects that are within few meters range of the Goggles under simulated orbital lighting conditions.
2. The Goggles **must** possess the computational capability to process the captured images using typical vision based navigation algorithms at a rate that is sufficient to control the SPHERES satellites.
3. The Goggles **must** provide a flexible software development environment that supports rapid development and iterative testing of a variety of algorithms.
4. The Goggles **must** provide the ability to reconfigure the optics hardware.

5. The Goggles **must** provide an interface for future hardware upgrades.

The first requirement is based on the need to test the algorithms in a laboratory environment. It inherently restricts this testbed to what is typically referred to as "close proximity" spacecraft operations, and excludes the development of algorithms for relative spacecraft operations at ranges between a few meters and a few kilometers. This limitation of scope is considered acceptable for the primary objective of the SPHERES Goggles. The second functional requirement is derived from the need to process the images using vision based navigation algorithms. The third, fourth and fifth functional requirements are based on the fact that the primary objective states that the SPHERES Goggles is a development, testing and maturation system, and not a flight hardware system. As a result, it should be as easy as possible to develop, test, modify and expand the SPHERES Goggles.

Using these functional requirements, a high-level system architecture can be defined.

2.2.4 System Architecture: Onboard versus Offboard Processing

In defining the system architecture, one fundamental tradeoff was heavily debated: How much processing should be done onboard, and how much processing should be done offboard? The primary objective and functional requirements indicate the need for flexibility in the algorithms and hardware to support research and development, however the constraints of SPHERES impose limitations on the amount of onboard processing power that can be physically attached to the SPHERES satellites. Two possible architectures that represent entirely onboard and entirely offboard processing architectures are shown in Figure 2-1.

On one hand, if all of the processing is done onboard in a 1 kg package, any algorithms developed will be immediately transferable to almost types of spacecraft. Additionally, developers will be forced to create the most efficient implementations of their algorithms.

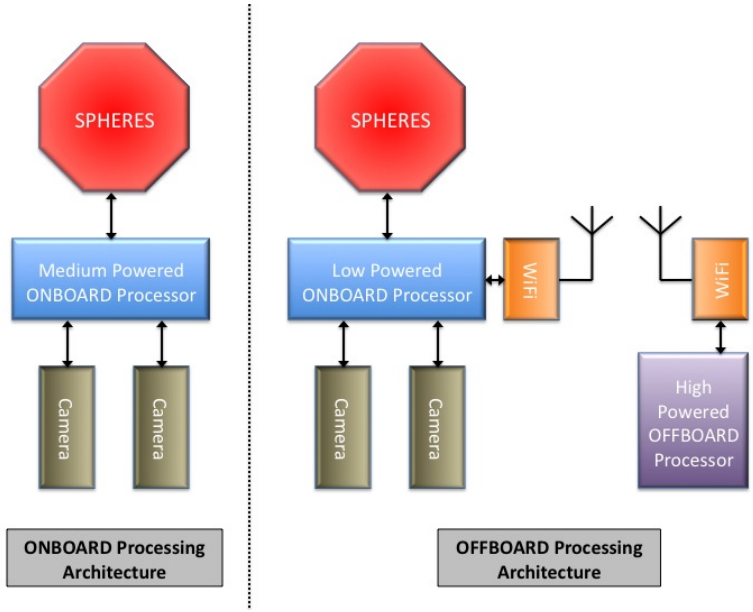


Figure 2-1: Goggles Architectures

On the other hand, the purpose of a testbed is to allow creativity and flexibility in the algorithms that are being developed. This would suggest that as much processing power as possible should be available to accommodate any possible type of algorithm. Additionally, many spacecraft will be capable of allowing more than 1 kg to be budgeted for the entire vision processing system.

The architecture that was selected was a compromise between the two previously mentioned extremes. The Goggles were required to be able to provide a reasonable amount of onboard processing power to execute typical embedded computer vision algorithms and in addition were required to be able to transfer the raw images over a wireless link to an offboard computer at a reasonable frame rate.

2.2.5 Subsystem Requirements

Using the functional requirements and the system architecture, a number of subsystem requirements are presented.

1. The Goggles **must** include at least two flight-traceable CMOS cameras that can be mounted in varying configurations (e.g. stereo and 45^o configurations).

(a) The cameras **must** provide 640×480 8-bit grayscale images at no less than 10 frames per second.

(b) The cameras **must** have software exposure control.

2. The Goggles **must** include onboard switchable fill lighting.

3. The Goggles **must** include at least one flight-traceable microprocessor capable of executing computer vision algorithms

   (a) The processor **must** run a Linux operating system

4. The Goggles **must** include an 802.11 wireless networking card that **should** capable of sending 10 frames per second from each onboard camera with lossless compression

5. The Goggles **must** include at wired expansion ports of at least one 100 MBps Ethernet connection and one USB 2.0 port.

The first and second subsystem requirement is derived from the first and second functional requirements, which specifies a need to image objects up to a few meters under simulated orbital lighting conditions at a frame-rate that allows for the control of the SPHERES satellites. A CMOS camera is specified rather than a CCD due to their inherent immunity to blooming under high contrast lighting that may occur on orbit. The 640×480 images with 8-bit grayscale resolution is selected as it provides the minimum file size while meeting the functional requirements for imaging range and quality. Although the SPHERES satellites are typically controlled using 5 Hz state estimates from the global estimator, the use of vision based navigation algorithms would require higher update rates due to the fact that small changes in attitude can result in very large changes in the image. As a result, the minimum frame-rate is inherently a function of the maximum angular velocity. However, the trade-off is that higher frame rates require more processing power. Therefore a minimum rate of 10 frames per second is selected as a compromise between processing power and maximum angular velocity.

The third subsystem requirement is based on the second and third functional requirement as well as the selected high-level system architecture. The selection of linux as the operating system for the Goggles is due to the wide availability of existing libraries and drivers for Linux when compared to other embedded operating systems (see Section 2.3.3 for further discussion). This enables the rapid development and testing that is specified in the third functional requirement. The fourth subsystem requirement is derived from the selected high level processing architecture, and is compatible with the first subsystem requirement. The fifth subsystem implements the electronic interface for future hardware upgrades that is specified by the fifth functional requirement. A USB and Ethernet port are specified since they provide a high data rate communications, and are commonly found on many currently available embedded electronics.

2.3 Goggles Design

2.3.1 Design Approach

The schedule and budget of the project required the Goggles to be designed in approximately eight months by one graduate student and one undergraduate student. Since the Goggles is a complex interconnected mechatronics system, it was designed, implemented and tested in an iterative process. The first step was to select the optics and electronics, and begin developing the software in a breadboard configuration. Once this was done, the electronics was attached to a SPHERE in an "un-integrated" fashion, similar to a flat satellite. In this stage all of the electronics would be fully connected, however they would be mechanically separated to make testing and debugging easier. After this was completed and tested, work began on the mechanical packaging to create a final Integrated SPHERES Goggles.

2.3.2 Optics and Electronics

Processor

The main driving factor for the electronics design was the Single Board Computer (SBC) selection. A number of options were considered according to a set of Figures of Merit (FOM) as detailed in Table 2.1. The processor type, power consumption and physical size FOMs are fairly self-explanatory, however the Floating Point and Instruction Set Architecture (ISA) should be clarified.

Table 2.1: Single Board Computer Comparison

Single Board Computer	**Processor**	**Thermal Design Power**	**Size**	**Instruction Set Architecture**	**Floating Point Unit**
Lippert CoreExpress	Intel Atom 1.6 GHz	5W	5.8cm×6.5cm	x86	Yes
Via Pico-ITX	Via C7 1.0 GHz	12W	10cm×7.2cm	x86	Yes
Texas Instruments Beagle Board	OMAP3530 (600 MHz Cortex-A8)	2W	7.6cm×7.6cm	ARM	Yes
InHand Fingertip5	XScale PXA320 806 MHz	<1W	6.1cm×8.6cm	ARM	No

There are a number of common instruction sets for embedded computers (x86, ARM, PowerPC etc.). For a research and development testbed, it is desirable to select an ISA that is very popular, and already has a widely available base of code. This is a distinctly different approach from a flight spacecraft where all of the software will be developed "in house" to ensure quality control and reliability.

Many embedded processors come without a hardware floating point unit (FPU), which is done to save power and space on the processor. In certain applications this is not a problem, as integer math can be used for most computations and floating point instructions can be emulated in software. However, in a number of vision based navigation algorithms, floating point computations are required (for example inverting a matrix in a Kalman Filter is best done with a hardware floating point

unit). Therefore, it is highly desirable to have a Floating Point Unit for the SPHERES Goggles, however it is not absolutely essential.

The processors listed in Table 2.1 met all of the requirements outlined in the previous sections (a number of potential SBCs were eliminated due to the fact that they did not have both USB 2.0 and 100 MBps Ethernet). One interesting observation is that most of the processors listed in the table below have been developed for smart-phones and tablet-PCs applications, items that did not exist in volume a few years ago. This implies that the aerospace industry can now leverage recent developments in the commercial smart-phone industry.

The processors are listed in order of how well they met the FOMs. Table 2.1 shows that the Lippert CoreExpress SBC was the clear winner in terms of the above FOMs. However, at the time of development, the Intel Atom was not widely available, and extra development time would be required for building additional printed circuit boards. Given that the SPHERES team also had previous experience with the Via Pico-ITX, it was selected as the SBC for the SPHERES Goggles. The SPHERES Goggles incorporated an 8 GB flash SATA drive as well as 1 GB of RAM. The Via Pico-ITX has 128 kB of cache memory, which is expected to limit the image processing performance, and is discussed further in Section 3.8.

Cameras

The selection of the cameras was a fairly simple process. The uEye LE cameras made by IDS Imaging were found to easily meet all of the requirements (an integrated frame buffer, USB interface and linux drivers). The specifications are summarized in Table 2.2. These cameras have a TTL flash output, which was used to synchronize the lights with the camera's exposure. A wide variety of lenses are available that fit into the M12 mount. Typically, a fairly wide-angle lenses is used with focal lengths between 2 and 5 mm.

Table 2.2: IDS Imaging uEye LE

Sensor	1/3" CMOS with Global Shutter
Camera Resolution	640 x 480 pixels
Lens Mount	S-Mount, M12
Frame Rate	87 FPS (Camera Max), 10 FPS (Typical)
Exposure	80 μ s - 5.5 s
Power Consumption	0.65 W each
Size	3.6cm×3.6cm×2.0cm
Mass	12 g

Fill Lights

Similar to the selection of the cameras, one set of lights was found that met all of the requirements and provided sufficient illumination in dark environments. The Luxeon III Star Light Emitting Diodes (LEDs) were selected due to their small size, high intensity light and moderate power consumption. Their specifications are shown in Table 2.3. A red-orange LED was selected because since this is the wavelength where the CMOS camera is the most sensitive. A collimator was also selected that focuses 85% of the light into a 50^o beam-width and is shown with the LED in Figure 2-2.

Table 2.3: Luxeon III Star Specifications

Model	Red-Orange Lambertian
Typical Luminous Flux	190 lm at 1400 mA
Typical Dominant Wavelength	617 nm (red-orange)
Typical Forward Voltage	2.95 V
Diameter	2 cm
Mass	5.5 g

In the optics design, Commercial Off-The-Shelf (COTS) LED drivers were used that regulated the LED current to 1000 mA, which corresponds to a typical power consumption of 2.95 W. Since there are 2 LEDs on the Goggles, the LEDs draw a significant amount of power. Therefore, a flash system was designed that only turns the LEDs on when the camera is capturing an image. Through tuning, it was found that sufficient illumination of objects a few meters away was possible with a 30ms exposure. Figure 2-3 shows two images captured by the cameras. The first image is

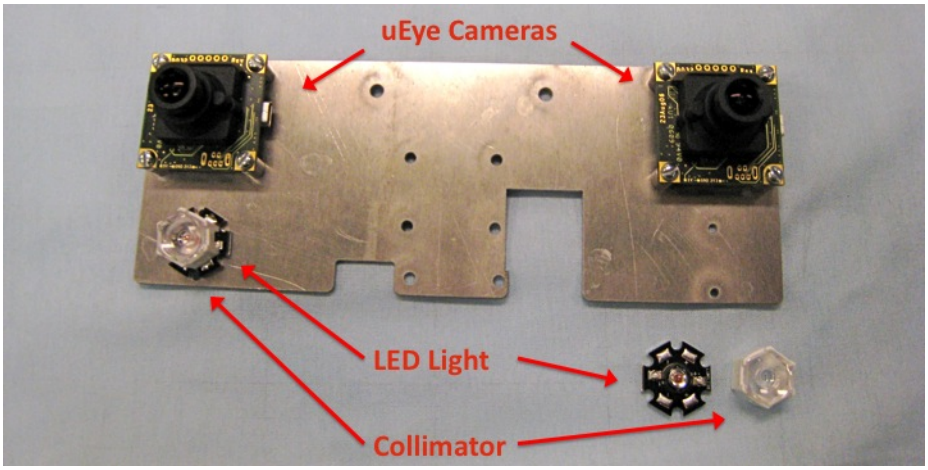
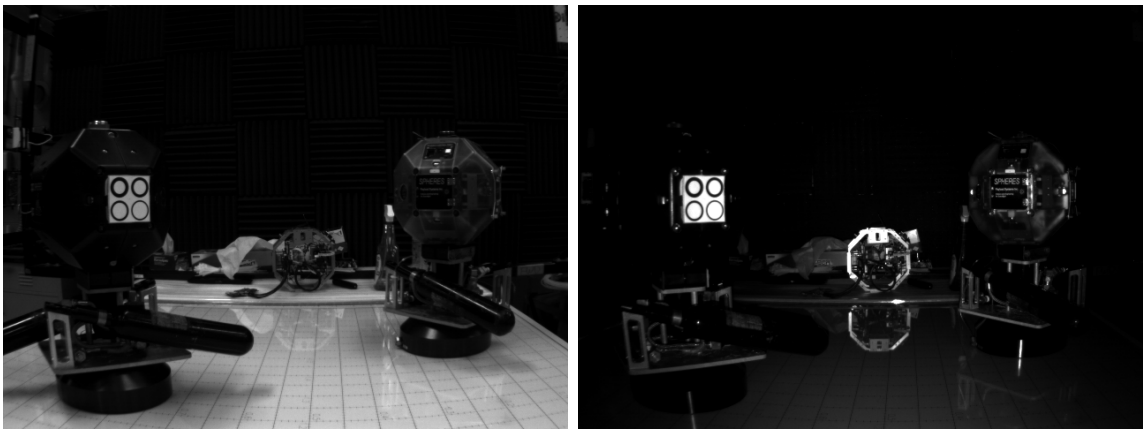


Figure 2-2: Goggles Optics



(a) Camera Image with Lights On

(b) Camera Image in Darkness (LED Fill Lights On)

Figure 2-3: Captured Camera Images

taken in the Space Systems Laboratory while the fluorescent room lights were turned on (without the use of the illuminating LEDs). In the second image, the fluorescent room lights were turned off, and the LED lights were used with a 30ms exposure (it should be noted that without the LEDs or the fluorescent room lights, the captured image would be completely black). These images show that the lights are sufficient for detecting an object, however the images will not provide much detail. Since the camera is capturing 10 frames per second, the LEDs operate at a 30% duty cycle. As a result, the net power consumption for two lights is approximately 1.8 Watts, and therefore the switching saves approximately 4.2 Watts.

Wireless Communications

Based on the subsystem requirements, the wireless communications system should be able to transmit 10 frames per second of two 640×480 frames at eight bits per pixel. In order to determine the data rate, it was assumed that the SPHERES Goggles would use the open source lossless compression software called Gzip. It was reasonable to assume a 50% compression ratio for each image. This requires the wireless communications system to be able to transmit approximately 23 mega-bits per second (Mbps), which is at the upper limit of the 802.11g standard's bandwidth (802.11g is designed for a maximum of 22 Mbps in each direction). It would be ideal to use an 802.11n network, which is capable of 72 Mbps in each direction, however at the time of development, these devices were not readily available in small packages, with linux device drivers. It was decided that a small reduction in image quality or frame-rate would be an acceptable tradeoff with the use of 802.11g. A QCom USB 2.0 device was selected and is specified in Table 2.4.

Table 2.4: QCom Device Characteristics

Mode	**Plug-In**	**FTP Transfer**	**FTP Receive**
Average Power	0.90 W	2.65 W	2.60 W
Maximum Power	2.15 W	2.95 W	2.85 W
Data Rate	N/A	20.97 MBps	18.04 MBps

Battery and Power Distribution

The power budget for all of the necessary components is listed in Table 2.5. The average power consumption that is used in the battery system design is 20.3 Watts.

Given that the SPHERES expansion port can only provide 17.5 W of power over 3 different voltages, it is not possible to power the Goggles using the SPHERES onboard batteries. This means an external battery must be used. In order to store as much energy as possible, while providing flight-traceability, a COTS integrated Lithium-Ion battery pack was selected. The details of the selected battery are specified in Table 2.6.

Table 2.5: Approximate Power Budget for Main Components

Item	**Average Power Consumption**
Pico-ITX	14.0 W
QCom Wireless Device	3.0 W
2 Cameras	1.3 W
LED Lights	2.0 W
Total	**20.3 W**

Table 2.6: Battery Specifications

Nominal Voltage	11.1 V (3 cells in Series)
Capacity	2500 mAh
Mass	154 g
Energy Density	180 Wh/kg
Dimensions	104cm×5.0cm×1.8cm
Built in Protection	Over-current, Over-voltage, Over-Drain, Short-Circuit, Polarity

With Lithium-Ion batteries there are a number of safety issues associated with over-draining the battery. For this reason, the battery selected comes with a protection circuit that will disconnect the battery before it can be damaged. However, if this were to occur it could cause damage to the Pico-ITX computer or the flash disk. Data may be lost or corrupted if there is any disk activity that is occurring when the battery disconnects. For this reason, we chose to incorporate a battery monitor that indicates the approximate voltage of the batteries and sounds a buzzer when the batteries are close to depleted.

In order to provide regulated power to the Pico-ITX board, a 20 Watt, 85% efficient, DC-DC converter was selected. This converter supplies power to the Pico-ITX and any devices that the Pico-ITX powers (e.g. Flash drive, powered USB devices). The 5V power supply for the USB cameras (1.3 Watts) and expansion port is provided by a linear regulator, which is built into the USB hub on the Pico-ITX. Since this regulator is only 45% efficient, a second DC-DC regulator that is 90% efficient was added to the Goggles design to provide power to the USB 802.11 device that consumes 3.0 Watts.

Final Goggles Electronic Design

In addition to the previously components, spare expansion ports are incorporated for future upgrades. These include a spare Ethernet, USB and unregulated 12V power line. Also, a connection for an external VGA monitor, keyboard and mouse is included so that the Goggles can be operated independently of the 802.11 and Ethernet connection.

Figure 2-4 shows the electronics architecture. Each of the blocks in the diagram represents a functional electronic component, while each of the lines represents a power connection, a data connection or both. The purpose of this diagram is to show the flow of power and data throughout the Goggles electronic components. The details of the implementation of this architecture (e.g. printed circuit boards, wiring and connectors) will be different between the Flat Goggles and the Integrated Goggles due to the differences in mechanical layout.

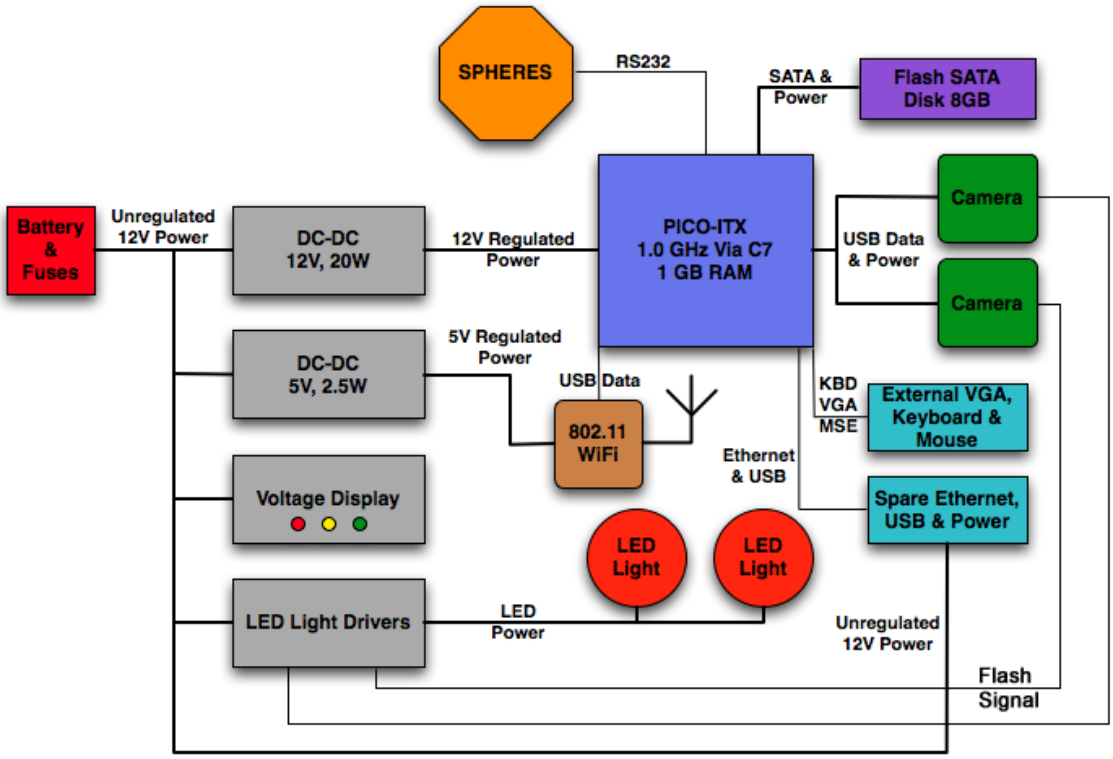


Figure 2-4: Electronics Architecture

2.3.3 Goggles Software

In the design of the SPHERES Goggles software architecture, an important distinction regarding software reliability must be made. The Goggles are not intended to have the reliability of a flight system, even though they must be flight-traceable. In many flight systems, a software failure would result in the loss of a significant amount of time, money and possibly even human life. Therefore every component of the software must be designed with the utmost care, attention to detail, testing and review. However, the SPHERES Goggles is a testbed system. Therefore the software architecture must enable rapid development of computer vision algorithms, which can be implemented, tested and evaluated in relatively short periods of time. In this type of a testbed system, ease of development may even be chosen at the expense of reliability. For this reason, a linux based operating system was selected over more reliable aerospace industry standard real-time operating systems such as VxWorks or QNX.

Ubuntu 8.04 "Hardy Heron" (based on the 2.6.24 kernel) was chosen as the particular linux distribution for the Goggles. This was due to the fact that Ubuntu is a widely adopted, well documented and designed to be as user friendly as possible. Additionally, the real-time kernel patches were incorporated to ensure proper performance. These patches are a critical element of an embedded real-time linux implementation since they enable kernel pre-emption, priority inheritance, high-resolution timers and convert all interrupt handlers to pre-emptible kernel threads.

Goggles Application Programmer Interface

In order to enable software development for the Goggles hardware, an Application Programmer Interface (API) was developed using the C programming language. The Goggles software API was designed to provide the functionality listed below. This functionality was divided into three independent APIs, the Goggles Optics API, the Goggles SPHERES API and the Goggles Network API. The functions of each of these APIs can be called from within the Goggles User Code, which is written and compiled as a separate executable program for each test.

1. Capture two frames from the camera and allow processing at a fixed interval (Goggles Optics API)

2. Send control commands to the SPHERE (Goggles SPHERES API)

3. Receive global metrology and gyroscope measurements from the SPHERE (Goggles SPHERES API)

4. Transfer images over the wireless network for off-board processing (Goggles Network API)

The dependencies of the API are shown in Figure 2-5 along with their relationship to existing linux libraries. The Goggles Optics, Goggles SPHERES and Goggles Network API blocks are shown in orange, while the Goggles User Code is shown in green, and all existing linux libraries that are used in the Goggles API are shown in red.

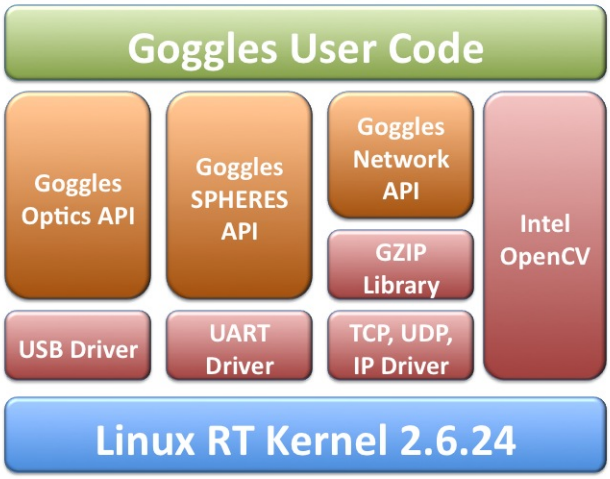


Figure 2-5: Goggles Software API

Algorithm 1 shows the pseudocode for the Goggles User Code. Inside a main while loop, an image is captured, a vision based navigation algorithm is called, actuation commands are sent to the SPHERES satellite and the images may optionally be transmitted over the network. It is important to note that the Optics API controls the timing of this loop. The image capture function in the Optics API will only

unblock the Goggles User Code and return an image after a hardware timer has expired on the camera. This hardware timer is set based on the desired frame rate that is specified during initialization (e.g. 10 Hz).

Algorithm 1 Psuedocode for Goggles User Code

```
Call Initialization Functions for Goggles APIs
while Test Running do
    Block until Images Returned (Goggles Optics API)
    Call Vision Based Navigation, and Control Algorithm (User provided code)
    Send Actuation Commands to SPHERES Satellite (Goggles SPHERES API)
    Send Images over Network (Optional - Goggles Network API)
end while
Call Closing Functions for Goggles API
```

The details of the API are described below, and further details on the SPHERES-Goggles interface can be found in Appendix A. The header files for the APIs and an example C program can be found in Appendix B.4.

The primary functions in the Optics API are shown in Table 2.7. This API configures the cameras to run at a specific frame rate based on their own internal clocks and will generate an interrupt when a new image is ready. The function captureTwoImages() will block until a new frame is captured and this interrupt is generated.

Table 2.7: Optics API Function

Function	**Description**
initTwoCameras()	This function initializes the cameras, sets frame rates, exposure, resolution and other parameters.
closeTwoCameras()	This function deallocates all structures for the cameras.
startTwoCameras()	This function starts the capturing process.
captureTwoImages()	This function blocks until a new image is captured, then returns a pointer to it.

The API for the SPHERES-Goggles interface is shown in Table 2.8. This API starts a seperate thread on initialization that is dedicated to receiving SPHERES state updates (global metrology) and gyroscope measurements. These values are time-stamped and stored in a global variable so that the control algorithms can access

them at a later time. One of the important operational aspects of the SPHERES-Goggles interface is that tests are started using the standard SPHERES Graphical User Interface (GUI). At the start of any Goggles program, waitGSPInitTest() must be called, which will block until it receives a start test message from the SPHERE. When the user stops a test using the SPHERES GUI, the SPHERE will send a stop test message to the Goggles, which is checked for by checkTestTerminate(). Control commands are sent by the sendCtrl() function which be specified in terms of targets in the inertial frame, or forces and torques in either the inertial or body frame. These commands are sent by the Goggles immediately and will be updated on the next control cycle of SPHERES.

Table 2.8: SPHERES API Function

Function	**Description**
initSPHERES()	This function initializes the SPHERES hardware including the RS232 port.
closeSPHERES()	This function deallocates all structures for the SPHERES interface.
waitGSPInitTest()	Blocks until a new SPHERES test is started.
checkTestTerminate()	Checks if a stop test message has been received.
sendCtrl()	This function sends control commands for the SPHERE to execute.

The API for the Networking interface is shown in Table 2.9. Initially a UDP transmission scheme was planned that had a single frame buffer, however it was found that many of the packets were reordered between frames. So rather than implementing a more complex multi-frame buffer, which would be the best solution, a simpler TCP protocol was selected whereby the entire frame was transmitted in a single packet (which the TCP driver can break up as needed).

This TCP-based Networking API implements a server-client architecture, where the Goggles act as a TCP server and another computer on the network (a "ground-station") acts as a TCP client. Once the ground-station connects to the Goggles, the Goggles may send images to the ground station. The protocol for sending the image involves a 4-byte header that specifies the number of bytes to expect in a packet that

Table 2.9: Networking API Function

Function	Description
initTCPNetworkSend()	This function initializes the system as the sender of images.
initTCPNetworkReceive()	This function initializes the system as the receiver of images.
closeNetwork()	Shuts down network.
tcpSendImage()	Sends the image using TCP.
tcpReceiveImage()	Blocks until an image is received.
downscaleImage()	Reduces the width and height of an image by a factor of 2.

is subsequently sent with the image data (there is currently only support for 320×240 and 640×480 pixel images).

Additionally, a compression scheme was planned based on the zlib (gzip) library. Experimentally, this scheme was able to obtain approximately 60% compression ratios for typical images, however it required a significant amount of processing time to perform this compression on the Goggles processor. This is likely due to the fact that zlib is trying to compress an entire image all at once. Due to the fact that the software is therefore trying to compress a 300 kB image while the Pico-ITX only has 128 kB of onboard cache, the long processing time is likely caused by a significant number of cache misses. As a result, image compression was implemented but disabled in the software. If a transmission scheme is improved so that each image can be broken into packets that are around 10 kB in size, this total compression time would likely be significantly less.

2.3.4 Flat Goggles Design

The first mechanical design of the Goggles is referred to as the Flat Goggles. This design placed all of the electronic components on a single flat aluminum plate attached to the SPHERES Air Carriage. The electronics were ultimately connected in their final configuration, however this flat layout allowed flexibility in debugging and development. This also enabled the separation of the development and testing of

the electronics from the problem of how to tightly package the electronics. The cameras, lights and SPHERES expansion port connectors were mounted on the SPHERE, rather than the flat plate, since their placement and orientation is important for testing. The components of the Flat Goggles are shown on a SPHERES air carriage in Figure 2-6 and 2-7.

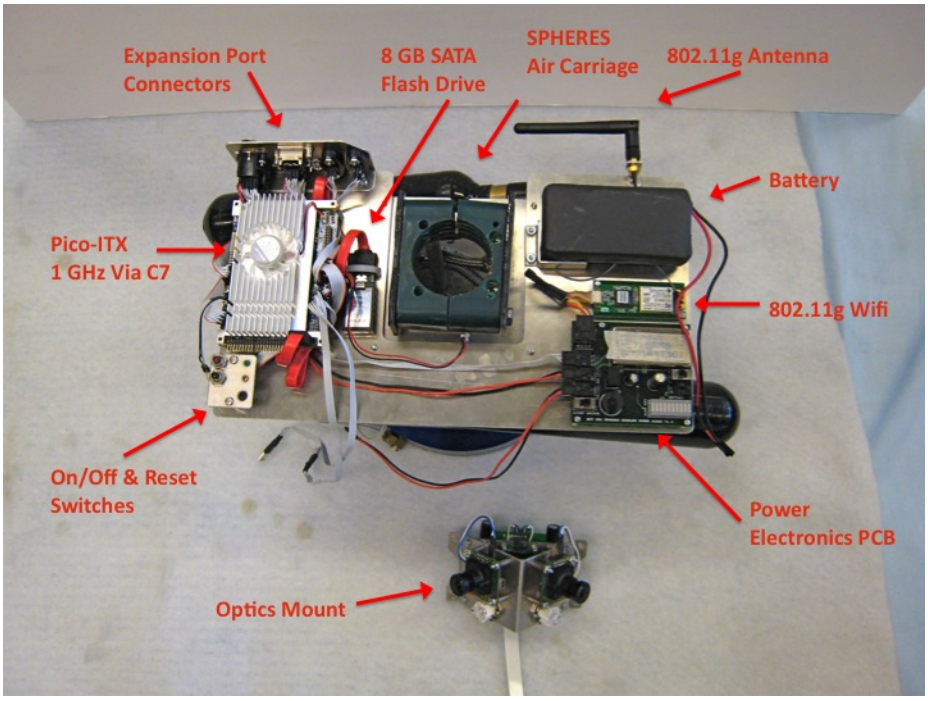


Figure 2-6: Top Down View of Flat Goggles on Air Carriage

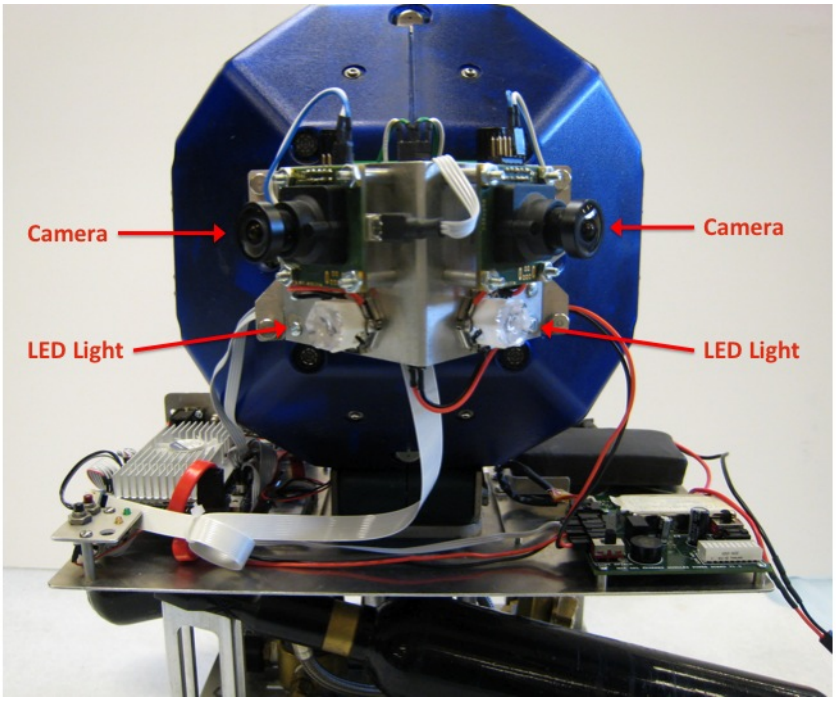


Figure 2-7: Front View of Flat Goggles mounted on SPHERES Satellite

2.3.5 Integrated Goggles Design

Once the Flat Goggles were built, verified and tested, the next step was to package the Goggles electronics in an integrated fashion so that they minimized the mass and volume of the Goggles package.

A number of key design decisions were made during the initial phases of integrating the Goggles. The first design decision that was made is that the Integrated Goggles should be separated into an Avionics Stack and an Optics Mount so that the optics can be interchanged. The Avionics Stack incorporates the Pico-ITX, battery, power electronics, expansion port connector and switches, while the Optics Mount consists of the cameras, LED lights and the spare Ethernet, USB and power connections. The interconnection between the Avionics Stack and Optics Mount is made with one electrical connector and four thumbnuts that connect to standoffs. The separation of components between the Avionics Stack and Optics Mount is shown in Figure 2-8.

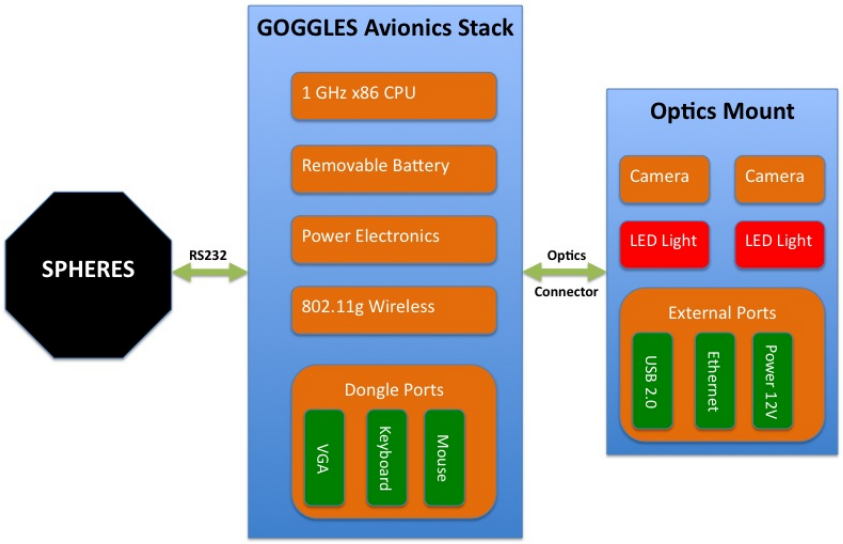


Figure 2-8: Separation of Electronic Components between Avionics Stack and Optics Mount

Another key decision was that the Goggles should be easy to assemble and disassemble. This was based lessons learned from building SPHERES; when something breaks inside a complex integrated system, being designed for serviceability can save

a lot of time and effort.

The third primary design decision was that the Avionics Stack should have a shell that is not structural or used for mounting any components. The shell's primary purpose is to provide the Goggles with protection from handling in a lab environment. This shell is manufactured using a 3-D Acrylonitrile Butadiene Styrene (ABS) plastic printer and was designed using a Computer Aided Design (CAD) program.

Figure 2-9 shows the final layout of the Integrated Goggles with the shell removed and the Goggles attached to SPHERES.

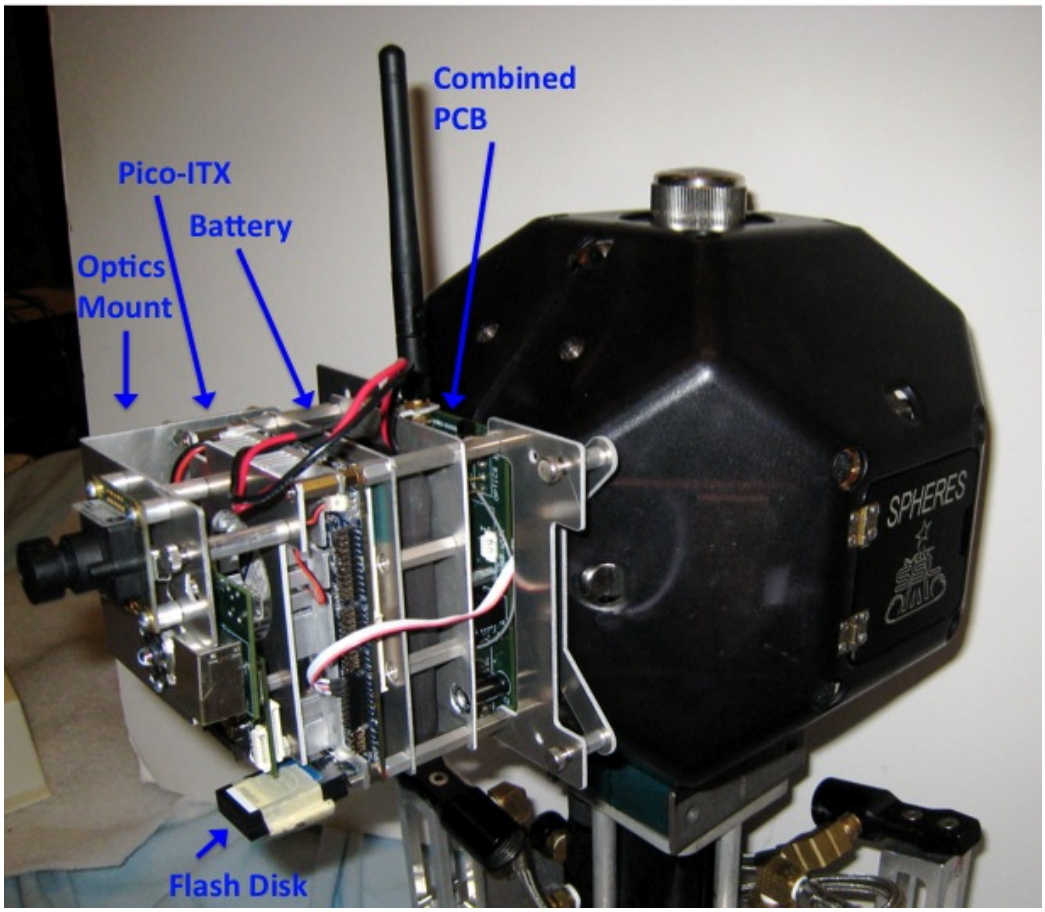


Figure 2-9: Fabricated Goggles

Figure 2-10 shows a dimensioned diagram of the Avionics Stack with the Shell attached. Figure 2-11 shows a dimensioned diagram of the Right Angle Optics Mount. Figures 2-12, 2-13 and 2-14 show the CAD design of the Integrated Goggles with dif-

ferent Optics Mount configurations. Photographs of the fabricated Integrated Goggles with the shell attached and right angle Optics Mount can be seen from multiple perspectives in Figures 2-15, 2-16, 2-17 and 2-18.

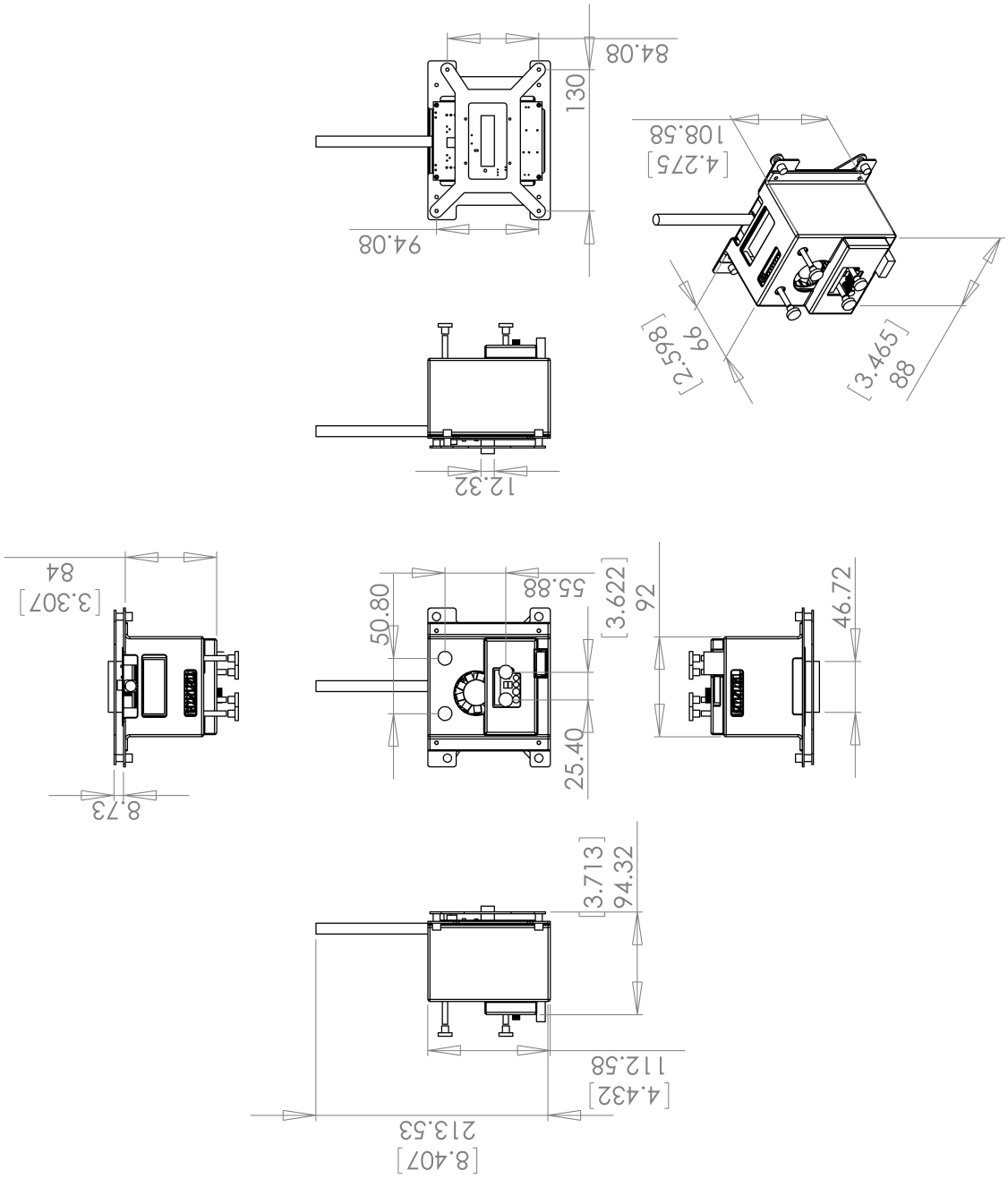


Figure 2-10: Dimensioned Diagram of Avionics Stack with Shell (units of millimeters [inches])

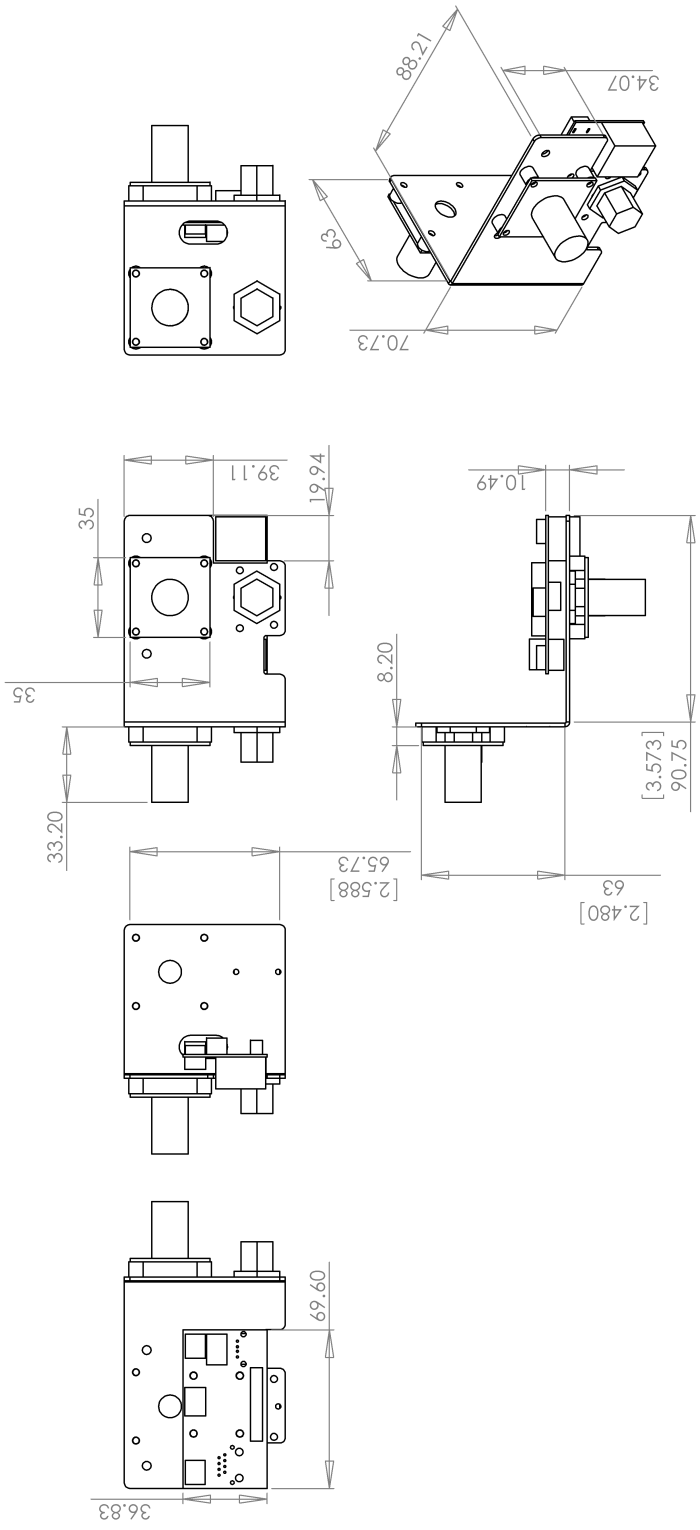


Figure 2-11: Dimensioned Diagram of Right Angle Optics Mount (units of millimeters [inches])

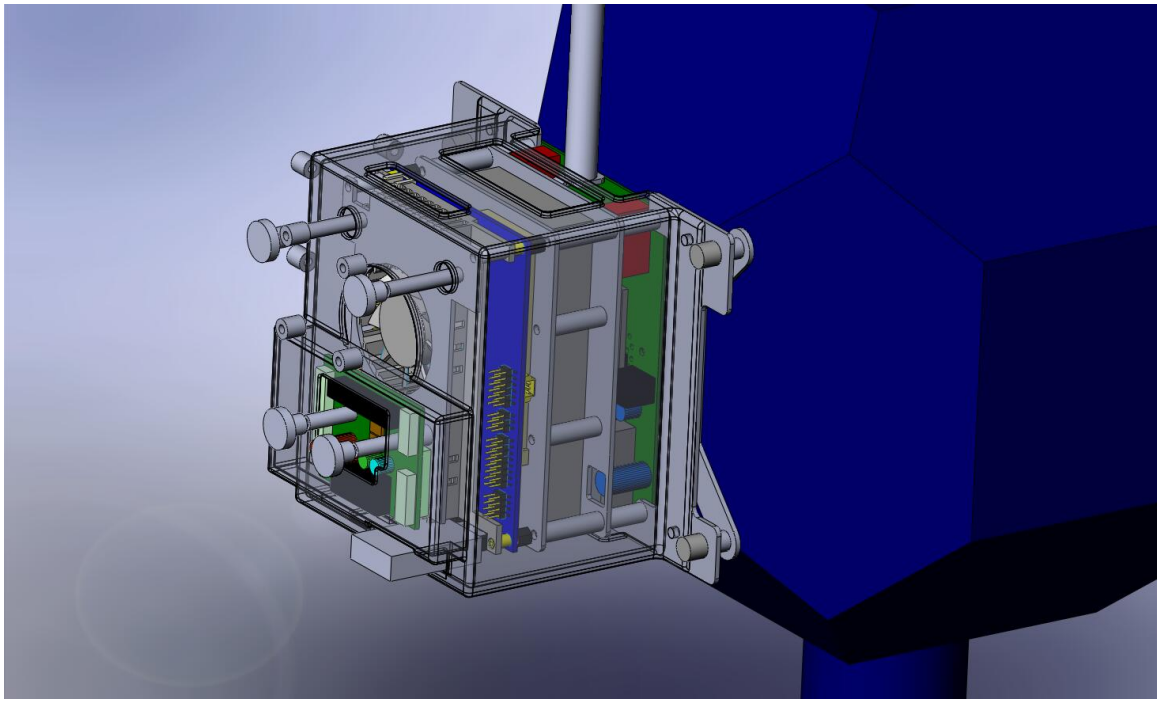


Figure 2-12: CAD Design of Avionics Stack

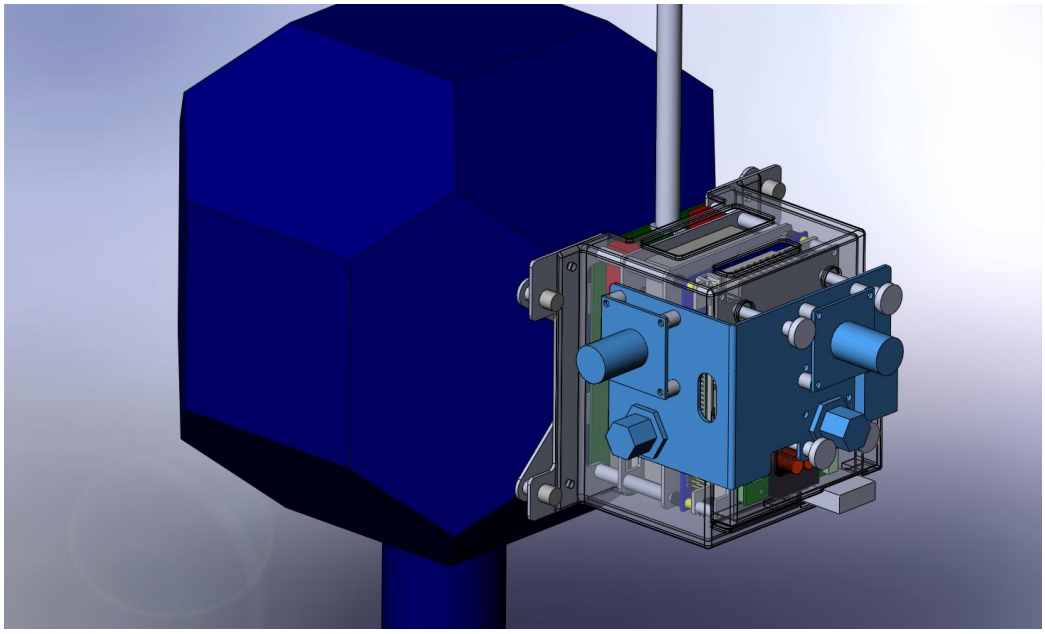


Figure 2-13: Integrated Goggles Design with Right Angle Optics

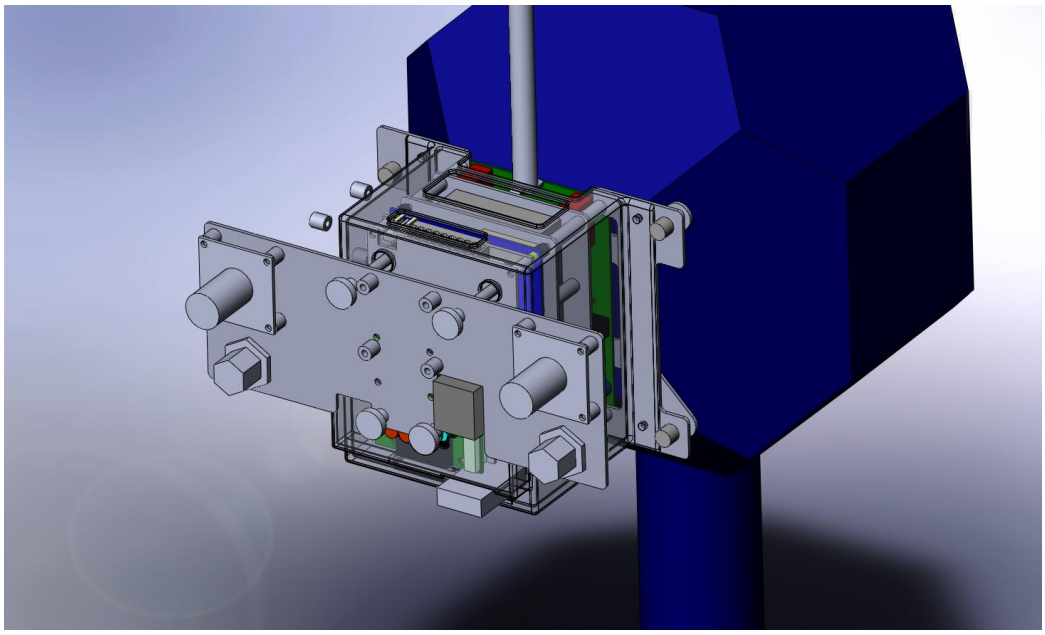


Figure 2-14: Stereo CAD Models of Integrated Goggles

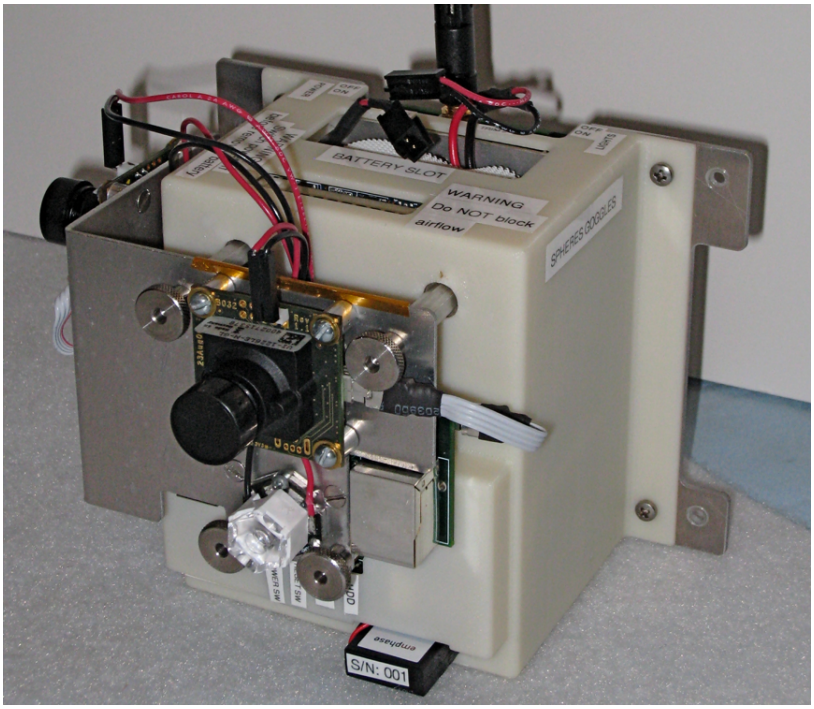


Figure 2-15: Front View of Integrated Goggles

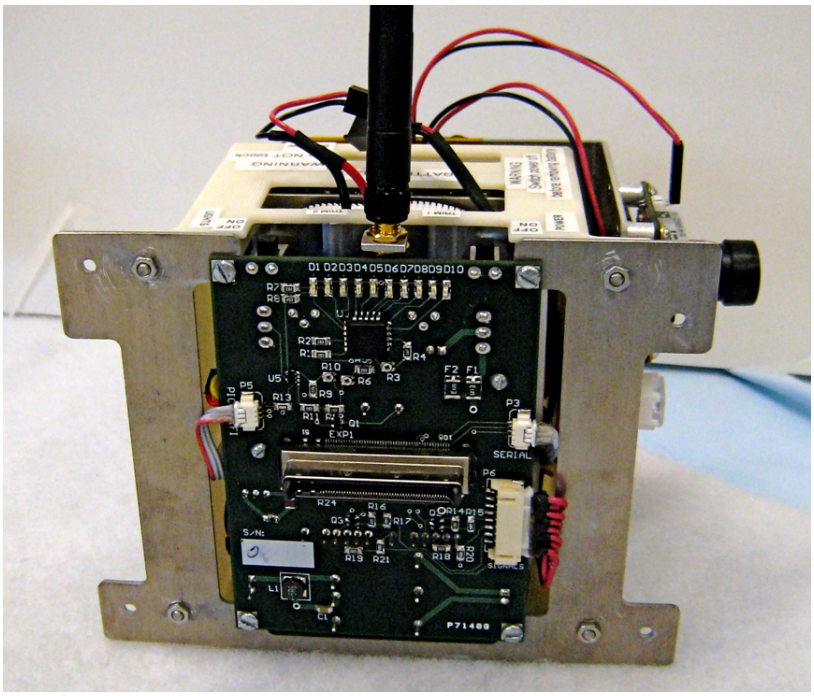


Figure 2-16: Rear View of Integrated Goggles

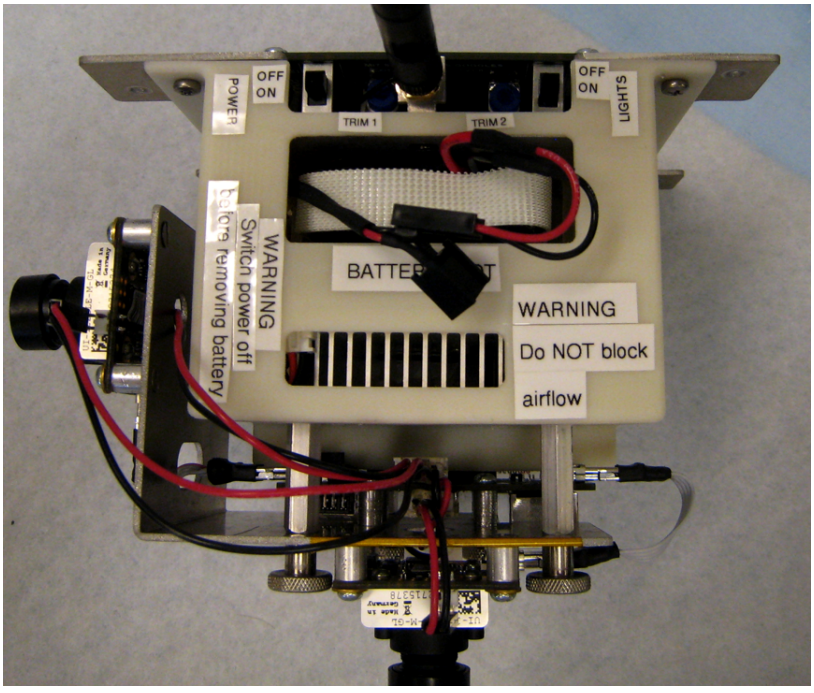


Figure 2-17: Top View of Integrated Goggles

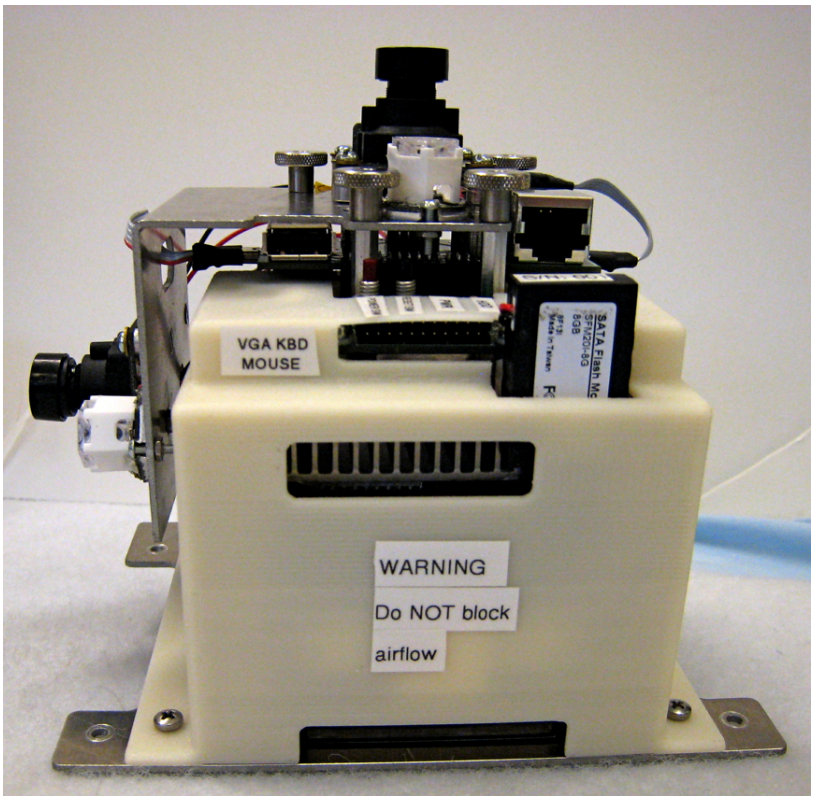


Figure 2-18: Bottom View of Integrated Goggles

One key aspect of the Integrated Goggles design was to ensure that there was sufficient cooling for the components inside the Avionics Stack. The Pico-ITX comes with a fan built into its heat sink, and is limited by a maximum temperature of 50^o Celsius. As a result, the Pico-ITX was placed at the outermost point of the Avionics Stack, and its exhaust ports were directed in the vertical direction (the direction of gravity), which prevents them from creating disturbance forces and torques (on the three degree of freedom table). The exhaust vents are visible in Figure 2-18.

Additionally, the 20 Watt (85% efficiency) DC-DC convertor often reaches a temperature of 60^o Celsius on the Flat Goggles platform. Although this is not a problem for the convertor, it was desirable to ensure it didn't overheat any further inside the Avionics Stack. This is because when the DC-DC converter reaches 75^o C it looses efficiency and at 100^o C the convertor is operating beyond its specifications. Therefore, the DC-DC convertor was thermally connected to the battery bracket and holes were designed into the shell to allow for natural convection, which are also visible in

Figures 2-17 and 2-18 (these holes are labelled with a "WARNING: Do NOT Block Air Vent" placard). If the SPHERES Goggles were to be operated on the inside of the ISS there would be a need to investigate and possibly redesign the thermal system for a microgravity environment.

The final mass of the Integrated Goggles is 615 grams without the battery and 895 grams with the battery. The maximum dimensions of the Integrated Goggles with the right angle Optics Mount are 130 mm by 109 mm by 66 mm.

2.4 Conclusions

This chapter has presented the systems engineering and design of an upgrade for the SPHERES Goggles. The detailed design of the electronics, optics, software and mechanical hardware was discussed for two versions of the hardware. Table 2.10 summarizes the final design of the Integrated Goggles.

It is anticipated that some redesign may be necessary to work in the International Space Station environment. For example, the lack of convective heat transfer in a microgravity environment may cause some cooling issues. Additionally, the use of lithium polymer batteries onboard the ISS may need to be reconsidered from a human safety perspective. It is suggested that moving to a processor with lower power consumption, such as the Intel Atom may alleviate some of these concerns. Lastly, the LED lights may also be problematic for the eyes of the astronauts as they can be painfully bright if viewed without eye protection. It is hoped that the Integrated Goggles will serve as a stepping-stone for a computer vision upgrade to SPHERES that is launched to the International Space Station in the near future.

Table 2.10: Integrated Goggles Specifications

Property	**Value**
Total Mass	895 g (with battery), 615 g (without battery)
Maximum Volume (Right Angle Optics Mount)	130 mm × 109 mm × 66 mm
Power Consumption	15 W (Idle), 18 W (Typical), 25 W (Max)
Processor	1 GHz Via C7, 128 kB L2 Cache
Chipset	VIA VX700
RAM	1GB DDR2 533 MHz
Flash Disk	8 GB SATA
Operating System	Real Time Ubuntu Linux 8.04 (Kernel 2.6.24-rt)
Cameras	2× IDS-Imaging uEye LE (1/3" CMOS with Global Shutter)
Camera Resolution	640 x 480 pixels
Lens Mount	S-Mount, M12
Frame Rate	87 FPS (Camera Max), 10 FPS (Typical)
Exposure	80 μ s - 5.5 s
Lights	2× Phillips Lumileds LXHL-LH3C (Red-Orange)
Lights Dominant Wavelength	617 nm
Lights Intensity	140 lm @ 2.9 W (per LED)
SPHERES-to-Goggles Communications	RS232 19.2 kbps
Wireless Communications	802.11g (54 Mbps)
Battery	Lithium Polymer 12V, 2.5Ah
External Ports	USB 2.0, Gigabit Ethernet, 12V Unregulated Power (2.0A Max)
Dongle Connector	Keyboard, Mouse and VGA

Chapter 3

Vision Based Relative Spacecraft Navigation using a Fiducial Marker

3.1 Motivation

Chapter 2 has described the design and development of a computer vision testbed (the SPHERES Goggles). This chapter will describe the first algorithm that was developed using this testbed. A visual navigation algorithm was implemented that utilizes a planar fiducial marker of known geometry to determine the relative position, orientation, linear and angular velocity between two spacecraft. This is obviously not the first system to offer this type of functionality on the ground or in space (see Section 1.3.2 for a review of prior work). There are two reasons why this type of system was selected as the first algorithm to be implemented on the SPHERES Goggles. The first reason is that the performance characteristics (accuracy, computational capability etc) of the SPHERES Goggles has not be experimentally characterized, and the use of well understood algorithms, such as fiducial based visual navigation, allows for reliable benchmarking of the Goggles. The second reason is that fiducial based visual navigation systems typically utilizes methods that are commonly found in algorithms for visual navigation in an unknown environment, which is currently an active topic of research. In this chapter, the details of the algorithm are discussed, and experimental results are presented using the SPHERES Goggles hardware to gauge the accuracy

and precision of the entire system.

3.2 Overview of Approach

3.2.1 Relative Pose Estimation Geometry

The fiducial target that will be used in this approach consists of concentric contrasting circles (see Section 3.4), which allows the navigation system to determine point correspondences. This algorithm does not incorporate any knowledge of forces and torques applied to either spacecraft, and does not incorporate any inertial measurements (although it could be extended to do so). The experimental setup using the SPHERES Goggles is shown in Figure 3-1.

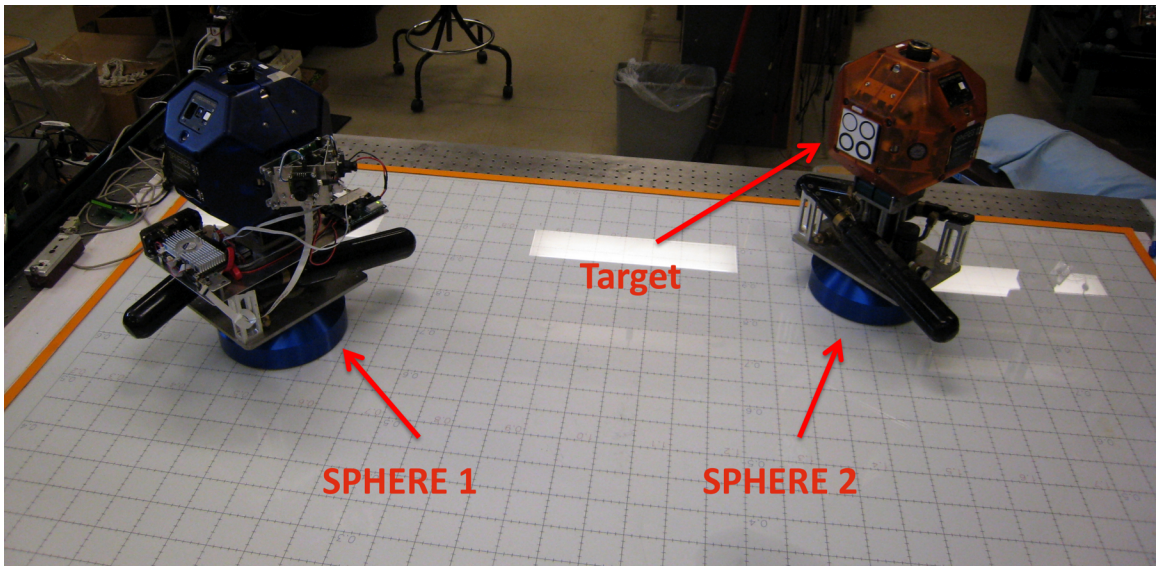


Figure 3-1: Relative Pose Estimation using SPHERES Goggles

It is useful to introduce notational conventions to represent coordinate transformations. Many texts develop these concepts and derive similar equations [31]. The three-by-one vector of real numbers $\mathbf{r}_{1/A}$ represents the location of point 1 in coordinate frame A. The three-by-three antisymmetric matrix of real numbers $\mathbf{R}_{BA}$ represents a rotation from coordinate frame A to coordinate frame B. See Appendix C.1 for details of rotational representations.

An example of the use of transformation of coordinate frames using the developed notation is shown in Equation 3.1. In this equation, $\mathbf{r}_{3/A}$ is Vector 3 in Frame A, $\mathbf{r}_{2/B}$ is Vector 2 in Frame B and $\mathbf{r}_{1/A}$ is Vector 1 in Frame A. Also, $\mathbf{R}_{AB}$ is the rotation matrix that transfers a vector from Frame B to Frame A. The vector sum of Vectors 1 and 2 is equal to vector 3. However vectors one and two are not available in the same coordinate frame, so the rotation matrix $\mathbf{R}_{AB}$ is used to transform the coordinate frames.

$$\mathbf{r}_{3/A} = \mathbf{R}_{AB}\mathbf{r}_{2/B} + \mathbf{r}_{1/A} \tag{3.1}$$

Using this notation, a system of coordinate frames is specified in Figure 3-1 for the relative navigation problem that is discussed in this chapter. Coordinate frames 1 and 2 are body-fixed frames attached to SPHERE 1 and 2 respectively. The coordinate frame C is the camera coordinate frame and the coordinate frame T is attached to the fiducial target that is mounted to the Velcro face of SPHERES 2. A top down view of the four coordinate frames, $\{1, 2, C, T\}$, their respective axes and the relative navigation geometry that is used in this approach is shown in Figure 3-2.

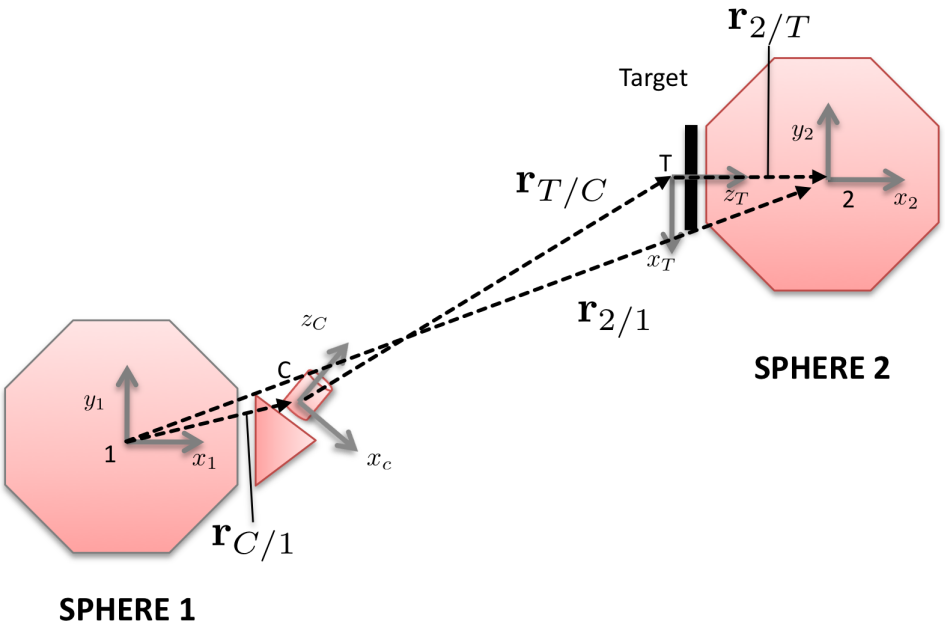


Figure 3-2: Top Down View of Relative Navigation Geometry and Coordinate Frames

In this approach, the relative pose will be defined as $\mathbf{r}_{2/1}$, the position of SPHERE 2 with respect to SPHERE 1, and $\mathbf{R}_{12}$ the rotation matrix from SPHERE 2 to SPHERE 1. It is assumed that both SPHERES move slowly with respect to their dynamics. Additionally, the values of $\{\mathbf{r}_{C/1}, \mathbf{r}_{2/T}, \mathbf{R}_{1C}, \mathbf{R}_{T2}\}$ specify the relative position and orientation between SPHERE 1 and the camera frame as well as between SPHERE 2 and the target frame. These values are known ahead of time and can be measured since these frames are mechanically attached.

The coordinate frame transformations are explicitly written below:

$$\mathbf{r}_{2/1} = \mathbf{r}_{C/1} + \mathbf{R}_{1C}\left(\mathbf{r}_{T/C} + \mathbf{R}_{CT}\mathbf{r}_{2/T}\right) \tag{3.2}$$

$$\mathbf{R}_{12} = \mathbf{R}_{1C}\mathbf{R}_{CT}\mathbf{R}_{T2} \tag{3.3}$$

All of the experiments in this approach used the Flat Goggles (see Figure 2-6). For the Flat Goggles, the values of the coordinate transformation parameters are as follows (all positions are measured in meters):

$$\mathbf{r}_{C/1} = \begin{bmatrix} 0.161 \\ 0.029 \\ 0.011 \end{bmatrix} \tag{3.4}$$

$$\mathbf{r}_{2/T} = \begin{bmatrix} 0.000 \\ 0.000 \\ 0.113 \end{bmatrix} \tag{3.5}$$

$$\mathbf{R}_{1C} = \begin{bmatrix} 0.7071 & 0.0000 & 0.7071 \\ -0.7071 & 0.0000 & 0.7071 \\ 0.0000 & -1.0000 & 0.0000 \end{bmatrix} \tag{3.6}$$

$$\mathbf{R}_{T2} = \begin{bmatrix} 0.0000 & -1.0000 & 0.0000 \\ 0.0000 & 0.0000 & -1.0000 \\ 1.0000 & 0.0000 & 0.0000 \end{bmatrix} \tag{3.7}$$

3.2.2 Relative Pose Estimation Algorithms

The algorithmic approach used for relative pose estimation in this chapter can be broken down into the following three steps. In order to implement these methods a calibrated camera will be required (Section 3.3.2).

1. Image processing to detect target's point correspondences (Section 3.4)
2. Non-linear iterative photogrammetric estimation of relative pose using a solution to exterior orientation (Section 3.5)
3. Filtering of relative pose using a Multiplicative Extended Kalman Filter (MEKF) (Section 3.6)

As was mentioned in the Section 1.3.2, relative pose estimation using computer vision has been extensively studied. As a result the above approach is a combination of a number of existing techniques. Step 1 is largely based on published work by Gatrell et. al. [28, 85]. Step 2 is an implementation of Haralick's iterative solution to the exterior orientation problem[38]. Step 3 is an implementation of a Multiplicative Extended Kalman Filter [59, 65, 18], which uses a state vector that consists of the relative position, velocity, orientation and angular velocity.

Current publications on visual navigation systems typically linearize the line of sight or collinearity equations (Equations 3.15 and 3.22) through the use of either an Extended Kalman Filter, a iterative nonlinear least-squares minimization algorithm or both[98, 51]. However both of these methods require an initial guess and may not converge to a global minimum for all possible initial conditions.

In the approach presented here, step 2 uses a solution to the exterior orientation problem that has been proved to be globally convergent for an infinite number of iterations. A proof is given by Haralick [38], that shows that the squared re-projection

error is monotonically decreasing for all possible initial conditions. This globally convergent property is important as it ensures that the Extended Kalman Filter update will not be given any outlier measurements that would lead to a non-Gaussian probability distribution of the state estimate (assuming the image processing system in step 1 does not incorrectly detect a target). It is important to note that although this algorithm is globally convergent for an infinite number of iterations, the number of iterations must be limited to a finite value, and as a result, there will be a non-zero error in the solution. Since it is well known that only one possible solution exists for four coplanar point correspondences, it is possible to specify an upper bound for the biased error in the solution. Section 3.9 presents upper bounds for this approach based on the experimental data presented in this chapter.

An additional benefit of this approach is that the exterior orientation directly solves for the position and orientation, which are states in the Extended Kalman Filter. As a result, the measurement model in the a posteriori update of the Kalman Filter is linear, which leads to significantly improved filter robustness (i.e. the only linearization that occurs is in the calculation of the a priori covariance matrix for the nonlinear attitude dynamics).

An interesting method was presented by Crassidis et. al. [17, 19] that the author was unaware of until after the work discussed here was complete. This method formulates a sequential nonlinear predictive filter to solve the relative pose estimation problem, and proves that the covariance matrix converges to the Cramér-Rao lower bound and is therefore an efficient estimator. It would be interesting to compare the results of Crassidis's estimator to the approach described here.

3.2.3 Contributions to Vision Based Navigation Algorithms

As was previously mentioned, neither the overall functionality of this system nor the individual algorithms that are used to implement each stage are unique to this thesis. However, the author is unaware of any previous publication the combines the Haralick's nonlinear "globally convergent" solution to the exterior orientation problem with a Multiplicative Extended Kalman Filter for the purpose of estimating

the relative states of two spacecraft. In applications where a large number of point correspondences are available, utilizing a linear method is typically not a problem. However, in this approach, only four point correspondences were available, which is the minimum number required to find a unique solution to the exterior orientation problem. In the case of four point correspondences, it was found that linear methods, when applied to noisy image processing measurements, would lead to very large errors that are unacceptable for any real-world implementation.

Additionally, the author has been unable to find a prior publication that explicitly describes a method for utilizing the Multiplicative Extended Kalman Filter to estimate angular velocity from orientation measurements. The mathematical details for using a MEKF to estimate angular velocity from orientation measurements are presented in Section 3.6. Typically, the filter is presented in a form where the orientation is estimated from a gyroscope's angular velocity measurements[18].

3.3 Camera Model and Calibration

3.3.1 Mathematical Model of a Pinhole Camera with Tangential and Radial Distortion

In this section a mathematical model of the camera is developed that is sufficient for the purpose of pose estimation. This approach is similar to the approach used in numerous texts [37, 45]. The starting point for this development is the pinhole perspective projection model, which is the most basic model that is used in photogrammetry. The primary assumption of the pinhole model is that the aperture of the camera is infinitesimally small. Additionally it is assumed that rays of light are not bent as they pass through the aperture and fall onto the image plane. It is important to note that the pinhole projection model possesses an infinite depth of field. Figure 3-3 shows a one dimensional illustration of a pinhole projection model.

In this diagram, an object of height x is located a distance of z away from a camera whose focal length is f (specified in meters). The image of the object that is projected

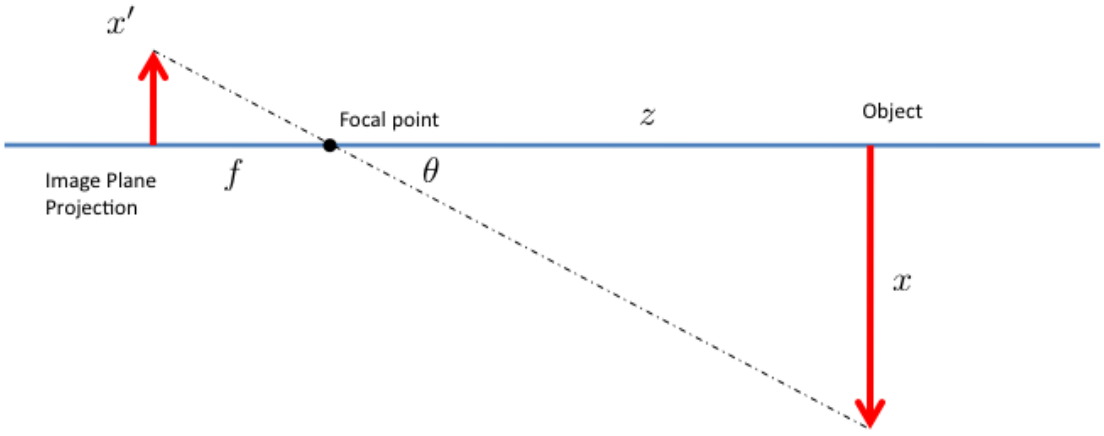


Figure 3-3: One Dimensional Pinhole Projection Model

onto the focal plane has a height of x'. Using similar triangles, an expression for the height of the image is derived:

$$x' = f\frac{x}{z} \tag{3.8}$$

Additionally, θ can be expressed as the line-of-sight angle from the camera's focal point to the tip of the object, using only the size of the image projection and the a priori knowledge of the focal length.

$$\theta = \arctan\left(\frac{x'}{f}\right) = \arctan\left(\frac{x}{z}\right) \tag{3.9}$$

The above formulas apply only to the projection of a two dimensional object onto a one dimensional plane. Using basic geometry, this can be extended to the projection of a three dimensional object onto a two dimensional plane ($[x', y']$). A scale factor s is introduced to represent the number of pixels per unit length. Also, the coordinates of the principle point are denoted by $[c_x, c_y]$ (measured in pixels), which removes the requirement that the origin of the camera frame be located at the principle point

thereby making the camera model more versatile. Now, the variables $[x'', y'']$ denote the location of the projection onto the image plane in units of pixels and $[\theta_a, \theta_e]$ represent the azimuth and elevation line-of-sight angles. Notice that these equations are nonlinear with respect to x, y and z.

$$x'' = sf\left(\frac{x}{z}\right) - c_x \tag{3.10}$$

$$y'' = sf\left(\frac{y}{z}\right) - c_y \tag{3.11}$$

$$\theta_a = \arctan\left(\frac{x'' - c_x}{sf}\right) = \arctan\left(\frac{x}{z}\right) \tag{3.12}$$

$$\theta_e = \arctan\left(\frac{y'' - c_y}{sf}\right) = \arctan\left(\frac{y}{\sqrt{x^2 + z^2}}\right) \tag{3.13}$$

Another commonly used parameterization is the unit line-of-sight-vector $[r_x, r_y, r_z]$. In the literature of visual spacecraft relative navigation (e.g. [98]), these equations are commonly linearized for the use with an Extended Kalman Filter. As was previously mentioned, this linearization will not be done in this approach, since it throws out important information.

$$\begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} = \frac{1}{\sqrt{x^2 + y^2 + z^2}} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.14}$$

$$= \frac{1}{\sqrt{x''^2 + y''^2 + f^2}} \begin{bmatrix} x'' \\ y'' \\ f \end{bmatrix} \tag{3.15}$$

$$= \begin{bmatrix} \cos\theta_e \sin\theta_a \\ \sin\theta_e \\ \cos\theta_e \cos\theta_a \end{bmatrix} \tag{3.16}$$

Up to this point, only one coordinate frame has been used; the camera coordinate frame. It is very useful to be able to relate the projected image coordinates to

coordinates in another coordinate frame. In doing this six degrees of freedom are introduced for the location and orientation of the camera with respect to the other coordinate frame.

Using the above notation for coordinate frame transformation, a mathematical model is described for the projection of a single point onto an image plane, where the camera is free to move. Figure 3-4 illustrates this situation.

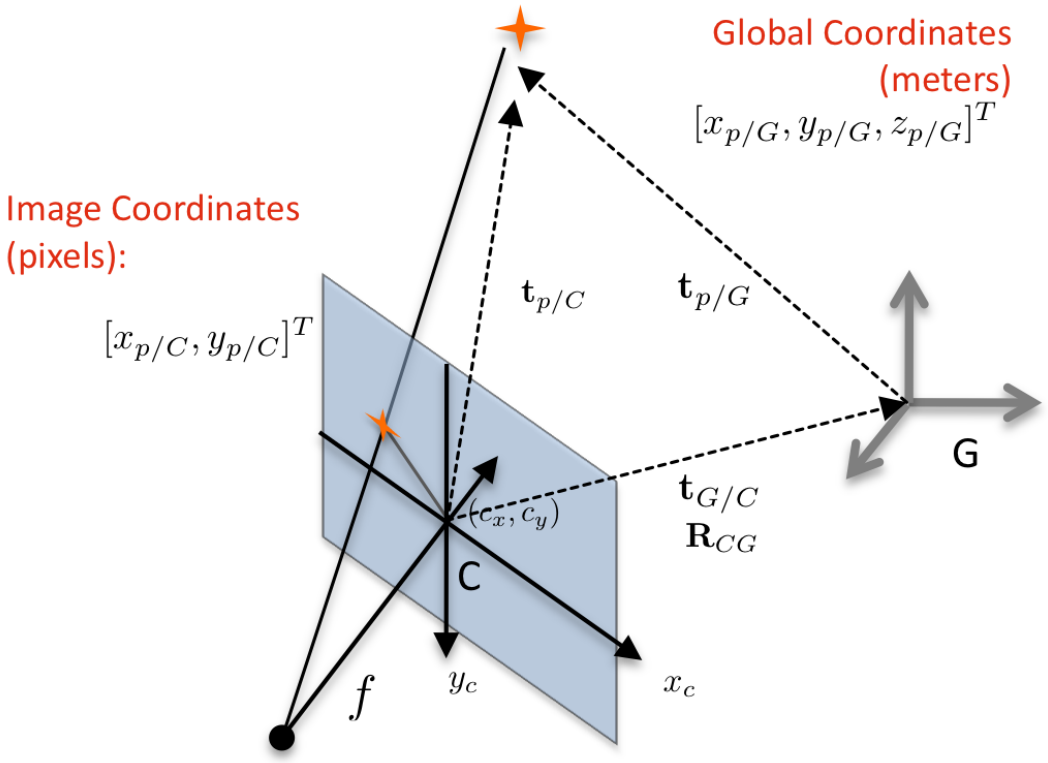


Figure 3-4: Projection of a Point onto an Image Plane

The coordinate frame C is the camera coordinate frame and is rigidly attached to the camera's image plane. The x and y-axis both lie on the plane of the image sensor, while the z-axis of the camera frame points away from the camera towards the object that it is imaging. The principal point in the camera plane is specified by $[c_x, c_y, 0]$ and the focal point is specified by $[c_x, c_y, -f]$.

The coordinate frame G is the global coordinate frame, which can move independently of the camera frame. The rotation from the global frame to the camera frame is given by $\mathbf{R}_{CG}$ and the translation from the origin of the camera frame to the origin

of the global frame is given by $\mathbf{t}_{G/C}$.

The location of a point p is known in the global frame coordinates, which is given by $[x_{p/G}, y_{p/G}, z_{p/G}]$ and is measured in meters (or another unit of length). The location of this same point in the camera coordinate frame is $\mathbf{t}_{p/C}$ and the transformation is given by the equation below.

$$\mathbf{t}_{p/C} = \mathbf{R}_{CG}\mathbf{t}_{p/G} + \mathbf{t}_{G/C} \tag{3.17}$$

The projection of this point onto the image plane is represented by $[x_{p/C}, y_{p/C}, z_{p/C}]$. In order to express the pinhole camera model using the global and camera coordinate frame, matrix notation and homogeneous coordinates are used [37, 9, 45]. Intermediate variables $[x_{p/C'}, y_{p/C'}, w_{p/C'}]^T$ are used for the homogeneous coordinates in the camera frame, where $w_{p/C'}$ is an additional parameter that represents a scale factor that will be divided out. The three dimensional coordinates of the point are represented by the homogeneous coordinates $[x_{p/G}, y_{p/G}, z_{p/G}, 1]^T$.

$$\begin{bmatrix} x_{p/C} \\ y_{p/C} \\ z_{p/C} \end{bmatrix} = \begin{bmatrix} \frac{x_{p/C'}}{w_{p/C'}} \\ \frac{y_{p/C'}}{w_{p/C'}} \\ 0 \end{bmatrix} \tag{3.18}$$

$$\begin{bmatrix} x_{p/C'} \\ y_{p/C'} \\ w_{p/C'} \end{bmatrix} = \begin{bmatrix} sf & 0 & c_x \\ 0 & sf & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{CG} & \mathbf{t}_{G/C} \end{bmatrix} \begin{bmatrix} x_{p/G} \\ y_{p/G} \\ z_{p/G} \\ 1 \end{bmatrix} \tag{3.19}$$

In the literature of photogrammetry, the first matrix on the right hand side of Equation 3.19 is referred to as the interior orientation, which describes the parameters specific to the camera. The second matrix is referred to as the exterior orientation, which describes the coordinate frame transformation between the global and camera frames.

Equations 3.19 and 3.18 can be expanded if the individual elements of the matrix

$\mathbf{R}_{CG}$ and translation vector $\mathbf{t}_{G/C}$ are parameterized.

$$\mathbf{R}_{CG} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{3.20}$$

$$\mathbf{t}_{G/C} = \begin{bmatrix} x_{G/C} \\ y_{G/C} \\ z_{G/C} \end{bmatrix} \tag{3.21}$$

Using the individual elements, Equations 3.19 and 3.18 are rewritten below.

$$x_{p/C} = sf\frac{r_{11}x_{p/G} + r_{12}y_{p/G} + r_{13}z_{p/G} + x_{G/C}}{r_{31}x_{p/G} + r_{32}y_{p/G} + r_{33}z_{p/G} + z_{G/C}} + c_x \tag{3.22}$$

$$y_{p/C} = sf\frac{r_{21}x_{p/G} + r_{22}y_{p/G} + r_{23}z_{p/G} + y_{G/C}}{r_{31}x_{p/G} + r_{32}y_{p/G} + r_{33}z_{p/G} + z_{G/C}} + c_y \tag{3.23}$$

In the literature of photogrammetry, Equations 3.22 and 3.23 are referred to as the collinearity equations. They fully describe the perspective projection of a three dimensional object onto a two dimensional image sensor using a perspective projection model. The above relationship utilizes a focal length, a scale factor (pixel size) and offset for the principle point of the image.

For the purposes of pose estimation, the only assumption that is currently unrealistic is that the aperture of the lens is infinitely small. In reality, the motion of the camera limits the exposure time of the sensor to prevent motion blur. Additionally, practical limitations on electronic amplifiers mean that the image signal cannot be increased without increasing the noise. As a result, it is necessary to increase the aperture of the lens to allow more photons to be collected on the image sensor. The tradeoff of this is that there is a finite depth of field.

Additionally, larger aperture lenses introduce distortions in the image. This can be extremely problematic for photogrammetric applications since it is assumed that

light travels along a straight line as it passes through the lens and falls onto the image sensor plane. As a result, these distortions must be corrected in software. In order to do this, a model of these distortions must be incorporated with the perspective projection equations that have been previously derived.

Prior work [10, 9] has shown that the most significant distortions for photogrammetric applications are tangential and radial distortions. Tangential distortions are primarily caused by a misalignment of the image sensor that causes it to not be perpendicular to the optical axis. Radial distortions, which are also known as barrel or pincushion distortions due to their appearance, are primarily caused by imperfect lens design and the introduction of stops into the optical path.

The uncorrected image plane coordinates that are found by applying the collinearity equations to a real-world (non-pinhole) camera are denoted as $[x^-_{p/C}, y^-_{p/C}]$. These are the coordinates prior to distortion correction. After the distortion correction is applied the image plane coordinates are specified by $[x^+_{p/C}, y^+_{p/C}]$. The model for the distortion is a linear shift in the pixel coordinates as shown below.

$$\begin{bmatrix} x^-_{p/C} \\ y^-_{p/C} \end{bmatrix} = \begin{bmatrix} x_{p/C} \\ y_{p/C} \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \end{bmatrix} \tag{3.24}$$

$$\begin{bmatrix} x^+_{p/C} \\ y^+_{p/C} \end{bmatrix} = \begin{bmatrix} x^-_{p/C} \\ y^-_{p/C} \end{bmatrix} + \delta_{tangential}(x^-_{p/C}, y^-_{p/C}) + \delta_{radial}(x^-_{p/C}, y^-_{p/C}) + \begin{bmatrix} c_x \\ c_y \end{bmatrix} \tag{3.25}$$

The radius from the optical center of a point, prior to distortion correction is represented by $r^-_{p/C}$. Additionally, the tangential distortion coefficients p_1, p_2 and the radial distortion coefficients k_1, k_2, k_3 are introduced to define the distortion model (see [10, 9] for details). In Section 3.3.2, the estimation of these parameters through the use of a calibration routine will be discussed.

$$r^-_{p/C} = \sqrt{(x^-_{p/C})^2 + (y^-_{p/C})^2} \tag{3.26}$$

$$\delta_{tangential}(x^-_{p/C}, y^-_{p/C}) = \begin{bmatrix} 2p_1 y^-_{p/C} + p_2(r^-_{p/C} + 2(x^-_{p/C})^2) \\ 2p_1(r^-/p/C + 2(y^-_{p/C})^2) + 2p_2 x^-_{p/C} \end{bmatrix} \tag{3.27}$$

$$\delta_{radial}(x^-_{p/C}, y^-_{p/C}) = \begin{bmatrix} x^-_{p/C}(k_1(r^-_{p/C})^2 + k_2(r^-_{p/C})^4 + k_3(r^-_{p/C})^6) \\ y^-_{p/C}(k_1(r^-_{p/C})^2 + k_2(r^-_{p/C})^4 + k_3(r^-_{p/C})^6) \end{bmatrix} \tag{3.28}$$

These equations fully define a non-linear camera model based on perspective projection that incorporates tangential and radial distortion. This model can be effectively summarized by a function $\mathbf{m}$. This function takes as its input a three dimensional point in the global frame, a set of parameters $\mathbf{\Theta}_I$ that are found by solving the interior orientation problem, and a set of parameters $\mathbf{\Theta}_E$ that are found by solving the exterior orientation problem.

$$\begin{bmatrix} x^+_{p/C} \\ y^+_{p/C} \end{bmatrix} = \mathbf{m}\left(\begin{bmatrix} x_{p/G} \\ y_{p/G} \\ z_{p/G} \end{bmatrix}, \mathbf{\Theta}_I, \mathbf{\Theta}_E \right) \tag{3.29}$$

$$\mathbf{\Theta}_I = \{sf, c_x, c_y, p_1, p_2, k_1, k_2, k_3\} \tag{3.30}$$

$$\mathbf{\Theta}_E = \left\{\mathbf{R}_{CG}, \mathbf{t}_{G/C}\right\} \tag{3.31}$$

3.3.2 Camera Calibration Using Tsai's Method and OpenCV

The camera intrinsic parameters $\{sf, c_x, c_y, p_1, p_2, k_1, k_2, k_3\}$ that were introduced in the previous section will not vary over time (assuming a zoom lens is not used). Therefore these parameters can be estimated offline and used as a-priori knowledge in the solution of real-time photogrammetric problems. Solving for these parameters is referred to as the interior orientation problem. A number of methods have been described to estimate these parameters [9, 43, 89, 102]. A brief overview of these methods is given here.

The first step in these methods is to collect a number of point correspondences that have accurately known coordinates in a global frame. This is typically done by imaging a chessboard pattern, which has been printed by a computer printer. A number of images are taken from different orientations and the corners of the chessboard are used as the known point corrsepondences. This results in m images of n coplanar corners on the chessboard. Images that were taken during the camera calibration process are shown in Figure 3-5. These images distinctly show the curvature of straight lines illustrating just how significant the radial distortion is for the optics on the SPHERES Goggles.

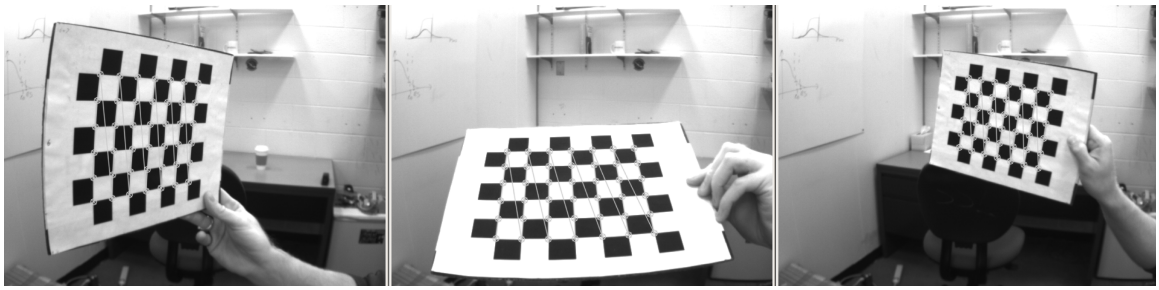


Figure 3-5: Images of Chessboard with Corners Labelled

The objective of the camera calibration algorithm is to determine a maximum likelihood estimate of $\mathbf{\Theta}_I = \{sf, c_x, c_y, p_1, p_2, k_1, k_2, k_3\}$ given the point correspondences of n corners in m images. This is equivalent to minimizing the a reprojection error over all points in all images [102]. In other words, $\forall k \in \mathbb{Z}^+ : k \leq m$:

$$\left\{\hat{\mathbf{\Theta}}_I, \hat{\mathbf{\Theta}}_{E,k}\right\} = \arg\min_{\mathbf{\Theta}_I, \mathbf{\Theta}_{E,k}} \sum_{i=1}^{n}\sum_{j=1}^{m} \left\| \begin{bmatrix} x_{i/C,j} \\ y_{i/C,j} \end{bmatrix} - \mathbf{m}\left(\begin{bmatrix} x_{i/G} \\ y_{i/G} \\ z_{i/G} \end{bmatrix}, \mathbf{\Theta}_I, \mathbf{\Theta}_{E,j} \right) \right\|^2 \tag{3.32}$$

The domain of minimization is over all real numbers, with the exception of sf which must be positive and that the rotation matrix must represent a valid rotation as specified by Equation C.6. This nonlinear function is typically minimized using a nonlinear least squares method such as the Levenberg-Marquardt algorithm, which is reviewed in Appendix C.4.

This method was implemented using the OpenCV library routines [9], by taking 20

images ($m = 20$) of a seven-by-eight chessboard grid ($n = 6 \times 7 = 42$). These images needed to be taken with varying positions and orientations to create a "rich" enough set of viewpoints. The functions *cvFindChesssboardCorners()*, *cvCalibrateCamera2()* and *cvUndistort2()* were respectively used to find the point correspondences, run the parameter estimation algorithm and remove the tangential and radial distortion. The values of the parameters that were found are listed in Table 3.1 with five significant digits (note that OpenCV assumes $k_3 = 0$).

Table 3.1: Calibration Coefficients

Parameter	**Value**
f	628.09 pixels (3.7686 mm)
c_x	321.98 pixels
c_y	208.165 pixels
k_1	-0.41956
k_2	0.18008
p_1	2.1562
p_2	-0.00046118

Most authors agree that an analytical error analysis is intractable for this high-dimensionality problem, however have found that experimentally that the focal length's standard deviation is less than 1% while the optical center's standard deviation is around one pixel [43, 102]. Since the projection equations are linear with respect to the intrinsic parameters, and given that 1 pixel corresponds to approximately 1.5 mm at a distance of one meter away, this is considered sufficient for the purpose of pose estimation.

3.4 Target Design and Detection

Since this relative pose estimation algorithm will utilize a fiducial marker of known geometry, one major design question is what should this fiducial marker look like. The image processing algorithms that are used to detect the target are among the most computationally expensive algorithms in a visual tracking system. As a result, the approach described here will focus on as simplistic a target as possible that can

be easily and reliably detected.

Fiducial targets typically use edges, lines or points as correspondences. Since this system will focus on the most simplistic method that will attempt to minimize the image processing requirements, point correspondences were selected. One of the most common types of point features to detect is a corner (this was used in the previous section for camera calibration). However, these methods typically break down in the presence of complex lighting and reflectance environments. Another alternative is Light Emitting Diode's, however this was not selected, as it would require the target vehicle to provide electrical power.

Research was undertaken by Gatrell et al. and Sklair et al. [28, 85], which developed a point target that is specifically designed for the lighting environments of space. The target they developed is referred to as concentric contrasting circles. A concentric contrasting circle is a target that has one small circle covering a larger circle, where the centroids of both circles are collocated. Figure 3-6 shows an image of a concentric contrasting circle, while Figure 3-7 shows the use of concentric contrasting circles on the International Space Station.

Figure 3-6: Concentric Contrasting Circle

There are a number of advantageous properties of contrasting circles that can be exploited to design a image processing algorithms for target recognition. The most important property is that the centroids of the circles remain collocated under rotations and translations. Additionally, the area ratio between the outer ring and inner circle will remain constant under rotation and translation. The collocated centroids are used as the point features for correspondence.

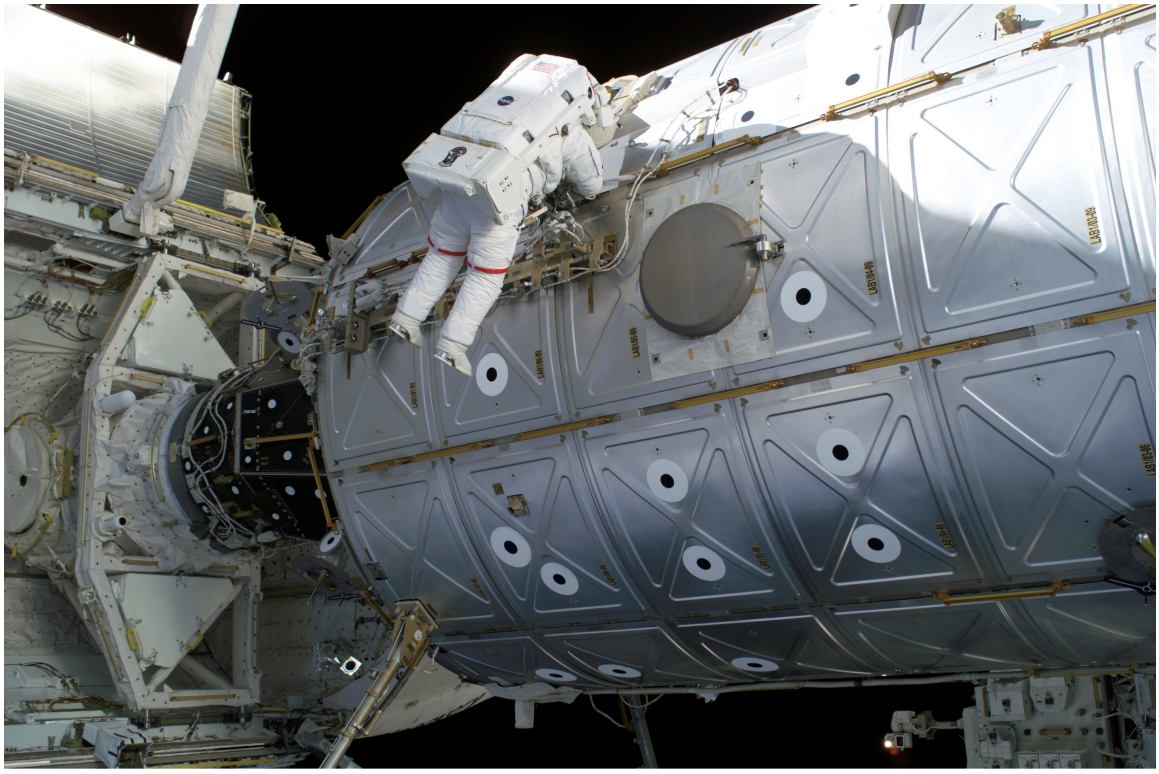


Figure 3-7: International Space Station with Concentric Contrasting Circles as Fiducial Markers [74]

A single point correspondence is insufficient to estimate the relative pose from a single image. Fischler and Bolles [26] have shown that at least four coplanar points are required to fully solve the exterior orientation problem with no ambiguities, however the more available point correspondences, the more accurate and robust the solution will be. However, the target design is constrained by the fact that it must be attached to the SPHERES satellites' Velcro face, and is therefore limited to seven cm by seven cm area. As a result, the minimum number of points was chosen so that the actual concentric circles could be as large as possible, and therefore visible from as far a range as possible. The resulting target is shown in Figure 3-8.

In order to facilitate correct correspondences, the diameter of the inner circle was varied to create different area ratios. The diameter of the outer circle is 2.8 centimeters, while the diameter of the inner circle is 2.4 cm, 2.2 cm, 2.0 cm and 1.8 cm. This creates area ratios of 0.1667, 0.2727, 0.4 and 0.5556 respectively. The separation between the centroids of each of the concentric circles is 3.2 cm in either

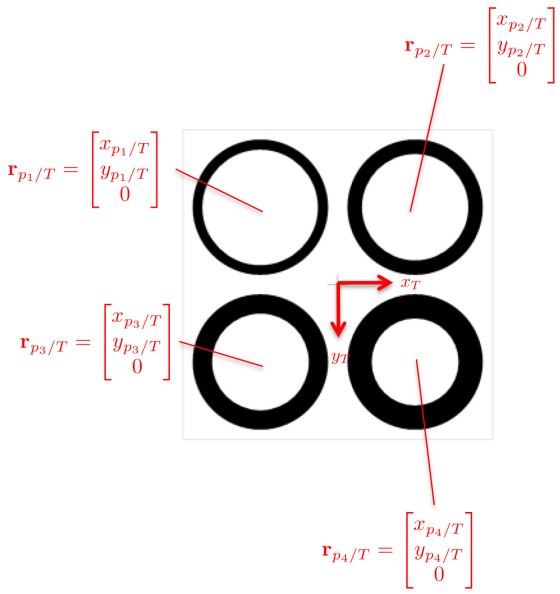


Figure 3-8: Coplanar Target of Four Contrasting Concentric Circles

the horizontal or vertical direction. A target coordinate frame T is defined with the x-axis and y-axis as shown in Figure 3-8, where the centroids of each of the four concentric circles is $[\pm 1.6, \pm 1.6, 0.0]$ cm.

In order to detect these targets, a seven step algorithm is used.

1. Image Segmentaton using Adaptive Thresholding
2. Connected Component Labelling
3. Filter Components by Area
4. Search for Collocated Centroids
5. Filter Components by Area Ratio and Colour
6. Four Point Check
7. Area Ratio Sorting and Correspondence

This algorithm is run on a sample image. Figure 3-9 shows the original image that is input to this algorithm (lens distortions have been removed in the manner

discussed in Section 3.3.2). The first two stages of this algorithm is illustrated in Figure 3-10 and 3-11.

Figure 3-9: Original Image

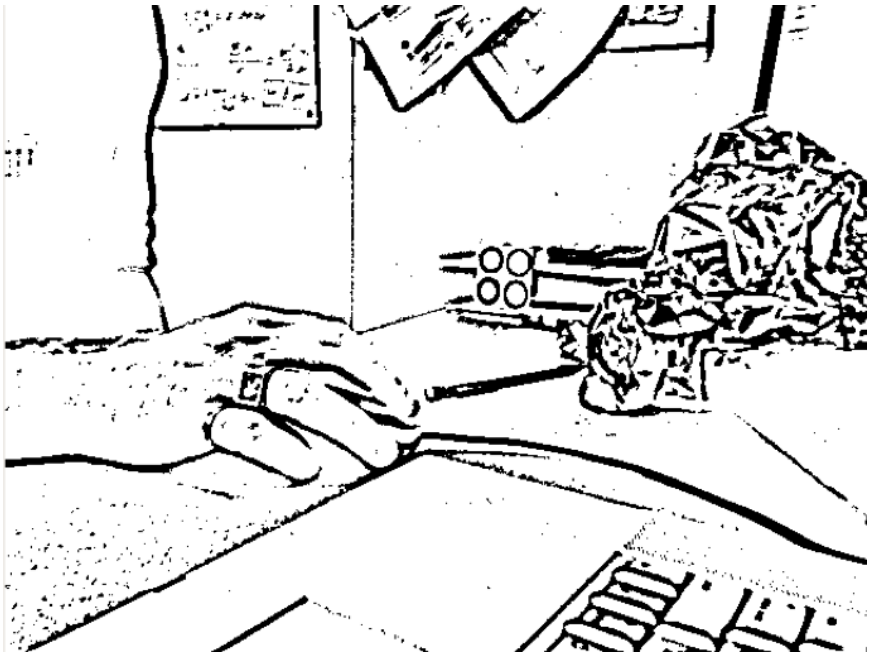


Figure 3-10: Step 1: Segmented Image

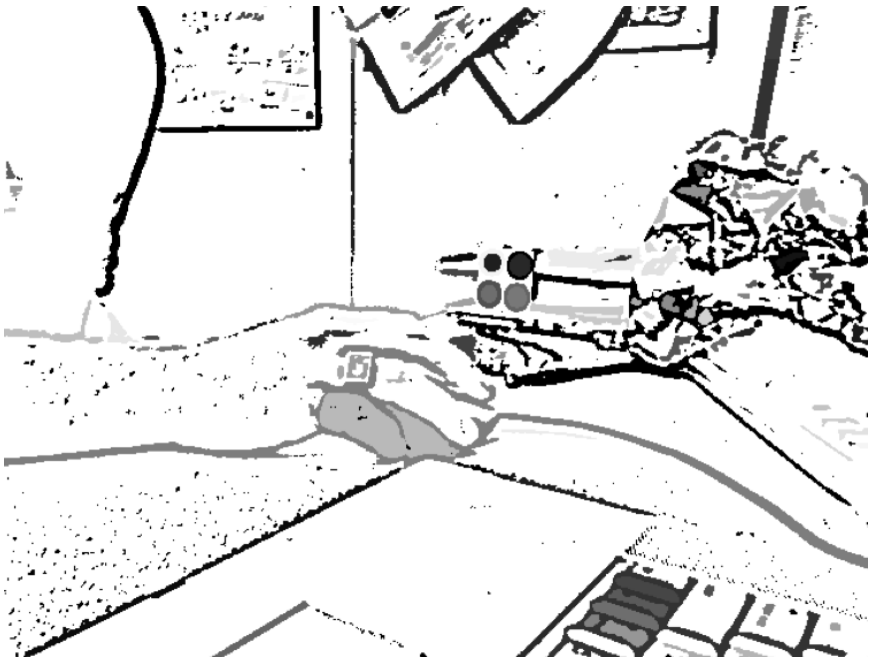


Figure 3-11: Step 2 & 3: Connected Component Labeling and Area Filtering

The segmentation algorithm uses an adaptive window threshold to segment the image into black and white. A basic segmentation algorithm that thresholds on a fixed value is not used due to the fact that variations in lighting conditions can cause entire areas of the image to segment to one colour or the other. An adaptive thresholding algorithm will compensate for variations in lighting conditions by using a threshold value that is a function of the local intensity mean.

The input to the threshold algorithm is an image $I(x, y)$ that has an intensity value at every x and y pixel location. The algorithm calculates a mean value over a window that has a height and width of $2w + 1$ pixels and is centered at the current x and y pixel location. This mean value is represented by $m(x, y)$. The output of this algorithm is $t(x, y)$ which is a threshold of the original image that is compared to the mean value minus an offset. This offset allows for the case where there is a constant intensity in the local averaging window. If no offset were used, all pixels in this window would be thresholded based only on the noise of the intensity of each pixel. This would create a "salt and pepper" noise texture that is an undesirable amplification of noise. Therefore the value of the threshold k should be set larger

than the value of the per pixel intensity noise.

$$m(x,y) = \frac{\sum_{j=y-w}^{y+w} \sum_{i=x-w}^{x+w} I(x,y)}{(2w+1)^2} \tag{3.33}$$

$$t(x,y) = \begin{cases} 255 & I(x,y) \geq m(x,y) - k \\ 0 & I(x,y) < m(x,y) - k \end{cases} \tag{3.34}$$

Connected component labeling is the process of assigning a single unique label to each region in a segmented image that is connected by adjacent pixels of the same colour . This is also referred to as blob labeling, and is typically performed by a sequential algorithm that classifies pixel values based on either the four or eight connected neighbours for each pixel (see Horn for details [45]). An open-source library, cvBlobsLib 6.1 [2] was used that implements the connected component algorithm and is designed to interface with the OpenCV API. This library also incorporates the ability to filter regions that are larger or smaller than a specified area. The minimum and maximum areas that were selected after experimental tuning were 20 pixels and 6000 pixels respectively. Figure 3-11 shows the results of the connected component labeling and area filtering, by colouring in regions that have been found with slightly different shades of grey. Notice that in Figure 3-11 there are a lot of regions that are coloured that are not concentric circles.

The fourth step of the algorithm searches for collocated centroids. Since the cvBlobsLib library outputs a list of regions (whose areas fall into the appropriate range) and provides the x and y component of the geometric centroid of each blob, a brute force search is performed to match regions whose centroids' fall within a specified radial distance of each other. Through experimental tuning, it was found that a radial distance of 1.5 pixels had very few false negatives and relatively few false positives.

The fifth step of the algorithm is based on the claim that the area ratio between the two concentric circles is invariant to shift and rotation [28, 85]. However, it was

found that there was a slight variation in the area ratio as the target becomes very small relative to the size of the pixels. This is due to the fact that when the edge of the target passes through a pixel, that pixel will not be counted as fractional unit of area, but will be counted as either no pixel or a full pixel. This is effectively a quantization error that causes slight shifts in the area ratio, since the adaptive filtering algorithm is more likely to expand the black circle and contract the white circle.

Using the fact that there is only a slight variation in area ratio over shifts and rotations of the target, the fifth step of the algorithm removes any pairs that do not have a larger black region and a smaller white region whose ratio of areas falls within a certain range. The range of the area ratio that was empirically determined for the targets was between 0.1 and 2.0.

The sixth step of the algorithm determines whether or not a target was seen in the image, by checking whether there are exactly four concentric regions remaining. If so, the algorithm concludes that a target has been found and proceeds to the seventh step. If not, the algorithm concludes that there is no target in the image, terminates and tells the Multiplicative Extended Kalman Filter that no target was found. Obviously this leaves the possibility for false positives (the target was not in the image but the algorithm reported finding the target) and false negatives (the target was in image and was not found). False negatives commonly occur when the target is either very far away, very close or at a large angle relative to the camera and the image processing algorithm does not have sufficient resolution to correctly detect the target. False positives can occur if another object in the background appears similar to a concentric circle target this can cause additional targets to be detected. For example, the SPHERES pressure gauge is occasionally mistaken by the algorithm as a target. If there are more than four targets found, subsets of the located targets could be tried in an approach that is similar to the Random Sample and Consensus algorithm [26]. However this type of approach was not selected due to the fact that it could introduce a significant amount of extra computation for the trial and error stages. Rather than performing the trial and error, the approach presented here throws out that image and tells the Multiplicative Extended Kalman Filter that no

target was found.

An extra step in the algorithm combats false positives due to outliers that look like concentric circles by incorporating information from a previous image into the current iteration. An additional filter is utilized in step one that removes all data outside of a circle that is centered on the origin of the target frame that was found in the previous image. As a result only concentric circles are found that are very close to the previous iteration's solution. Given that it is not expected that a satellite will move significantly between frames, this is a valid assumption.

To summarize, this section has presented the design of a fiducial marker based on concentric contrasting circles that are used as point correspondences between the image plane and a known location of the circles in the frame of the target. The next step in the overall pose estimation algorithm is to use these point correspondences to estimate the rotation and translation between the target and camera frame. This is referred to as the exterior orientation problem.

3.5 Solution of Exterior Orientation

The problem of exterior orientation (which is also known as resection) is to estimate the rotation and translation between a camera and a fixed coordinate frame, given a set of two-dimensional points on the image plane and the corresponding three-dimensional coordinates in the fixed coordinate frame. This is a problem as old as photogrammetry itself[34, 71, 45], as there are literally hundreds of published solutions to this problem ranging from the German mathematician Grunert (reviewed in English by Haralick [39]) to a recent publication by Fiore in 2001[25]. The primary reason for this is because there still does not exist a closed form (i.e. non-iterative) solution for the n-point least-squares estimate of the exterior orientation that does not linearize or approximate the problem in some way.

The geometry of a three-point exterior orientation problem is shown in Figure 3-12. The geometry of this problem can be viewed as a tetrahedron. The green lines $\{d_1, d_2, d_3\}$ form the base of a tetrahedron, which is known based on the target's

geometry, while the red lines $\{S_1, S_2, S_3\}$ for the sides of the tetrahedron, which meet at the focal point of the camera. The angles between the faces of the tetrahedron $\{\alpha_1, \alpha_2, \alpha_3\}$ are known based on a priori knowledge of the focal length and the location of the corresponding points in the camera frame $[x_{p_i/C}, y_{p_i/C}]^T$.

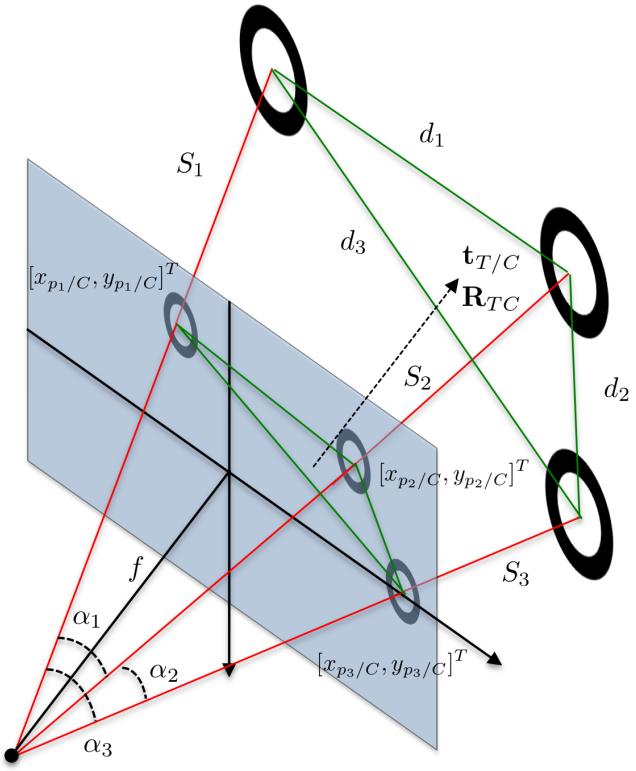


Figure 3-12: Three Point Exterior Orientation Problem

The problem of exterior orientation can be restated as the problem of determining the lengths of the sides of a tetrahedron $\{S_1, S_2, S_3\}$, given the dimensions of the base, $\{d_1, d_2, d_3\}$, and the angles between each side$\{\alpha_1, \alpha_2, \alpha_3\}$. The line-of-sight vectors are defined as j_i in Equation 3.35. The angles between the line-of-sight vectors are α_i.

$$j_i = \frac{1}{\sqrt{x_{p_i/C}^2 + y_{p_i/C}^2 + f^2}} \begin{bmatrix} x_{p_i/C} \\ y_{p_i/C} \\ f \end{bmatrix} \tag{3.35}$$

$$\cos\alpha_1 = j_1 \cdot j_2 \tag{3.36}$$

$$\cos\alpha_2 = j_2 \cdot j_3 \tag{3.37}$$

$$\cos\alpha_3 = j_1 \cdot j_3 \tag{3.38}$$

$$\tag{3.39}$$

Using the cosine law, three non-linear equations (Equations 3.40, 3.41 and 3.42) can be solved for the three unknowns $\{S_1, S_2, S_3\}$. A number of closed form solutions are available which transform these equations in to a fourth order polynomial whose root must be found (an in depth review is given by Haralick [39]).

$$d_1 = S_1^2 + S_2^2 - 2S_1S_2\cos\alpha_1 \tag{3.40}$$

$$d_2 = S_2^2 + S_3^2 - 2S_2S_3\cos\alpha_2 \tag{3.41}$$

$$d_3 = S_1^2 + S_3^2 - 2S_1S_3\cos\alpha_3 \tag{3.42}$$

$$\tag{3.43}$$

In general, there are four solutions to this polynomial equation, which correspond to four different geometric configurations. Fischler and Bolles identified this problem [26] and showed that there is no way to determine which of these four solutions is correct. In order to find a solution to the exterior orientation problem without ambiguity, a minimum of four coplanar or six non-coplanar points are required. As was discussed previously, there is no closed form least squares solution to the exterior orientation problem for more than three points.

In this approach, an iterative method is used to solve the exterior orientation problem, which was proposed and shown to be monotonically globally convergent by Haralick (this is described as *Method 2* in Section IV. in [38]). Figure 3-13 illustrates the method for a four point coplanar correspondence.

Given that there are N target points (in Figure 3-13 $N = 4$), for each point $n : 1 \leq n \leq N$, the location of these points in the fixed frame is $\mathbf{y}_n$, while the three dimensional coordinates of these points in the camera frame is defined as $\mathbf{x}_n$.

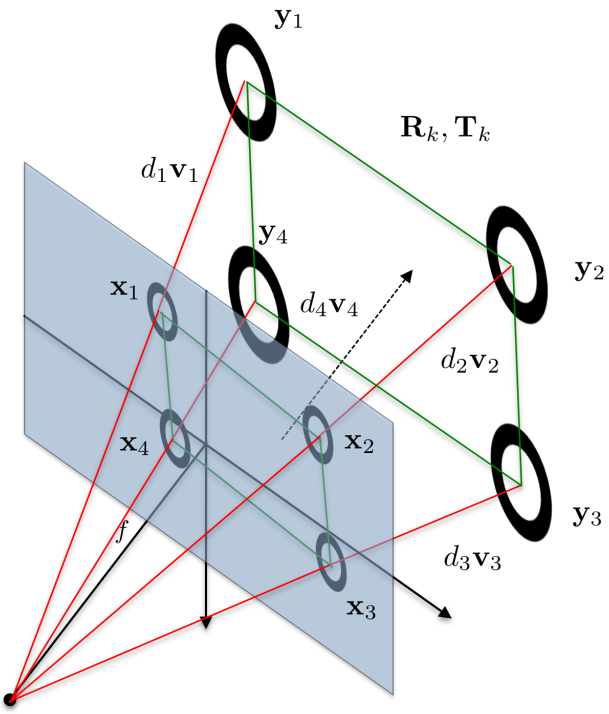


Figure 3-13: Four-Point Correspondence Geometrey of Haralick's Iterative Nonlinear Solution to Exterior Orientation

The coordinates are related using the rotation $\mathbf{R}$ and translation $\mathbf{T}$ between the two frames. The vector $[u_{n1}, u_{n2}]$ are defined as the image coordinates of the target points.

$$\mathbf{x}_n = \mathbf{R}\mathbf{y}_n + \mathbf{T} \tag{3.44}$$

The vector $d_n\mathbf{v}_n$ represents the length of each of the sides of the square pyramid, where d_n is an scalar and $\mathbf{v}_n$ is defined based on the image coordinates $\mathbf{x}_n$ and the camera's focal length. Note that in Equation 3.45, u_{n1},u_{n2} and f must have the same units (either pixels or metric units).

$$\mathbf{v}_n = \begin{bmatrix} \frac{u_{n1}}{f} \\ \frac{u_{n1}}{f} \\ 1 \end{bmatrix} \tag{3.45}$$

Using the above notation, the exterior orientation problem can be formulated as a least squares minimization problem that will be minimized over k iterations. An exact solution to the exterior orientation will find that:

$$\begin{aligned} \mathbf{x}_n &= d_n \mathbf{v}_n && (3.46) \\ \mathbf{R}\mathbf{y}_n + \mathbf{T} &= d_n \mathbf{v}_n && (3.47) \\ \Rightarrow \mathbf{R}\mathbf{y}_n + \mathbf{T} - d_n \mathbf{v}_n &= 0 && (3.48) \end{aligned}$$

The mean squared reprojection error (at any given iteration of the algorithm) is defined by ϵ_k^2. Note that the $\frac{1}{n}$ proportionality constant has been dropped.

$$\epsilon_k^2 = \sum \left|\left|\mathbf{R}_k \mathbf{y}_n + \mathbf{T}_k - d_n^k \mathbf{v}_n\right|\right|^2 \tag{3.49}$$

As a result, the exterior orientation problem solves for the values of d_n that minimize the mean squared error. This is given by Equation 3.50.

$$\{d_n^*\} = \arg\min_{d_n} \sum_{n=1}^{N} ||\mathbf{R}_k \mathbf{y}_n + \mathbf{T}_k - d_n \mathbf{v}_n||^2 \tag{3.50}$$

The algorithmic solution for this minimization problem is to iteratively refine the estimates of d_n, using a two step process at each step k in the iteration. The first step is to find an value of the rotation and translation that minimizes the mean squared error reprojection error using the current estimate d_n^k. This problem is shown in Equation 3.51, which is the absolute orientation problem. The closed form solution to this problem that is used is described in Section 3.5.1.

$$\{\mathbf{R}_k^*, \mathbf{T}_k^*\} = \arg\min_{\mathbf{R}_k, \mathbf{T}_k} \sum_{n=1}^{N} \left|\left|d_n^k \mathbf{v}_n - (\mathbf{R}_k \mathbf{y}_n + \mathbf{T}_k)\right|\right|^2 \tag{3.51}$$

The second step in each iteration is to determine a new estimate d_n^{k+1} of the

scale factors. The method used for this is given in Equation 3.52. This is the scalar projection of the new value best guess for $\mathbf{R}_k\mathbf{y}_n + \mathbf{T}_k = \mathbf{x}_n^k$.

$$d_n^{k+1} = \frac{(\mathbf{R}_k\mathbf{y}_n + \mathbf{T}_k)^T\mathbf{v}_n}{\mathbf{v}_n^T\mathbf{v}_n} \tag{3.52}$$

To review, the iterative exterior orientation algorithm is as follows:

1. Determine initial guess for scale factors d_n^0 and define maximal allowable change in mean squared error (i.e. stopping threshold) $\Delta\epsilon_{MAX}^2$.
2. Fix the value of d_n^k and solve absolute orientation problem in Equation 3.51 using methods of Section 3.5.1.
3. Update value of d_n^{k+1} using Equation 3.52
4. Compute the mean squared error ϵ_{k+1}^2 using Equation 3.49.
5. If $\epsilon_k^2 - \epsilon_{k+1}^2 < \Delta\epsilon_{MAX}^2$, stop algorithm and return values of $\mathbf{R}_k, \mathbf{T}_k$, otherwise return to Step 2.

The values of $\mathbf{R}_k, \mathbf{T}_k$ returned by this algorithm are by definition equal to $\mathbf{R}_{CT}, \mathbf{T}_{T/C}$ that is given in Section 3.2.

Haralick showed that this algorithm is globally convergent for an infinite number of iterations. In other words, given any starting condition, $\epsilon_{k+1}^2 \leq \epsilon_k^2$. Figure 3-14 plots the square root of the 2-norm of the error for each of target points as well as the root mean squared error for all of the target points. Experimental data was used where pixel coordinates of the target locations were given with four significant digits. The initial starting conditions for all values of d_n^0 were given as 0.5 meters. The figure shows that this error converges to 0.2 millimeters, in approximately 3000 iterations. Additionally, the difference in the total squared error, given by $\epsilon_k^2 - \epsilon_{k+1}^2$ is plotted in Figure 3-15. This figure clearly shows that the error is monotonically decreasing until the error approaches the limit of precision of the computer at approximately 10000 iterations. However, given the real-time constraints of the system, the maximum number of iterations is fixed. This means that there will always be a bias in the result

of the exterior orientation algorithm, but the bias will be bounded if the reprojection error is less than a fixed threshold.

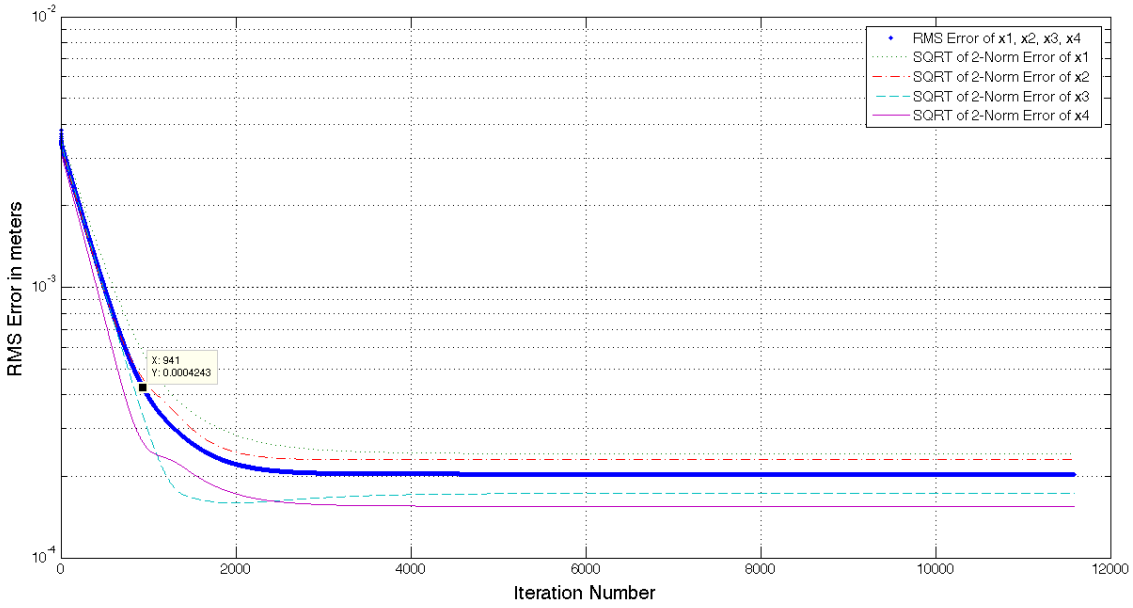


Figure 3-14: Error of Exterior Orientation Algorithm for each Iteration

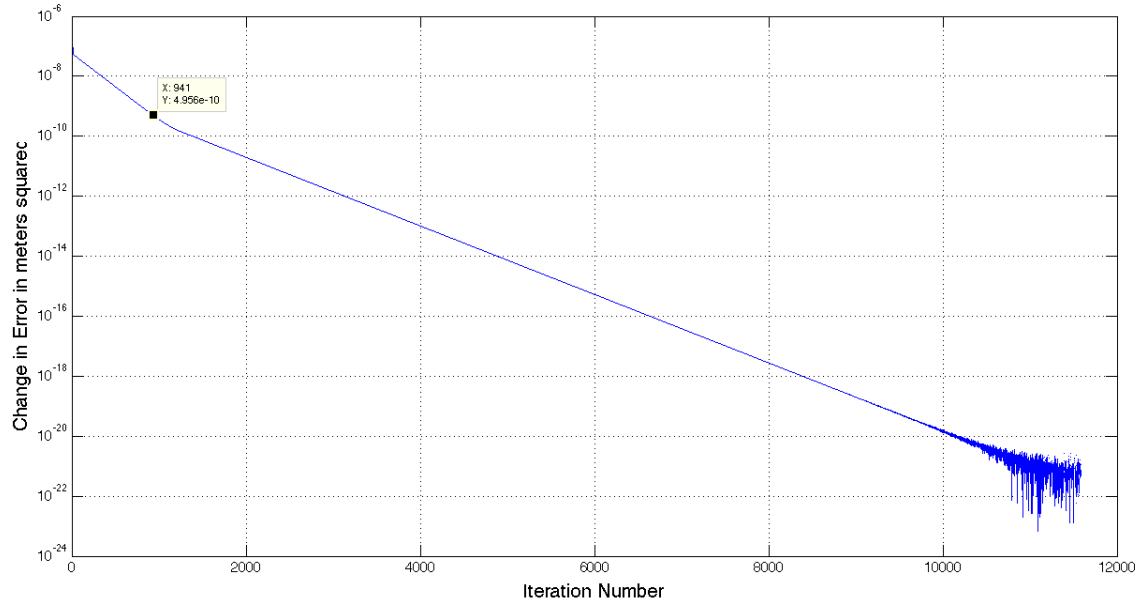


Figure 3-15: Difference in Total Squared Error for each Iteration: $\epsilon_k^2 - \epsilon_{k+1}^2$

In selecting the stopping threshold $\Delta\epsilon_{MAX}^2$, sub-millimeter precision was desired. Examining Figure 3-14 shows that after 941 iterations the error was 0.4243 millime-

ters. The change in the total squared error as shown in Figure 3-15 at 941 iterations is $4.956e-10$. As a result a stopping threshold was set to $\Delta\epsilon^2_{MAX} = 5.0e-10$.

Given that this algorithm is to be implemented on a real time system, the fact that this iterative solution requires nearly 1000 steps to converge is problematic (especially since a singular value decomposition is required in each step). It was experimentally found that using the values of d_n found in the previous iteration as the initial conditions could reduce the number of steps required to less than 30, provided the target did not move more than a few centimeters. Additionally, a second stopping condition was used that limited the number of iterations that could be run in a single cycle of the visual pose estimation algorithm to 1000. If this was reached, the nonlinear minimization would be stopped and its values of d_n would be used as the initial conditions in the next time step of the pose estimation algorithm. Typically, it will take multiple time steps of the pose estimation algorithm to converge on initialization, but once the solution has converged, the exterior orientation solution will typically be solved in less than 30 iterations.

The source code for the solution to the exterior orientation problem is given in Appendix B.1.

3.5.1 Solution to Absolute Orientation using Singular Value Decomposition

The absolute orientation problem is another fundamental photogrammetric problem that is very similar to the exterior orientation problem. However, unlike the exterior orientation problem, a number of closed for least squares solutions have been found [5, 44, 82]. The solution in this section follows Arun's method[5].

Consider the three-point correspondence tetrahedron discussed in the previous section and diagrammed in Figure 3-12. In absolute orientation it is assumed that the lengths of the sides of the tetrahedron $\{S_1, S_2, S_3\}$ as well as the geometry of the base of the tetrahedron $\{d_1, d_2, d_3\}$ is given. The objective is to compute the rotation and translation between the camera and target frame $\mathbf{R}$ and $\mathbf{T}$.

In a manner identical to the exterior orientation problem, $\mathbf{y}_n$ is defined and $\mathbf{x}_n$ is now a known quantity.

$$\mathbf{x}_n = \mathbf{R}\mathbf{y}_n + \mathbf{T} \tag{3.53}$$

The least square reprojection error that must be minimized is

$$\{\mathbf{R}^*, \mathbf{T}^*\} = \arg\min_{\mathbf{R},\mathbf{T}} \sum_{n=1}^{N} ||\mathbf{x}_n - (\mathbf{R}\mathbf{y}_n + \mathbf{T})||^2 \tag{3.54}$$

A key observation is that the rotation and translation can be solved independently if the locations of the target points are measured from the centroid of all of the targets in both frames. Therefore the centroids are defined as $\bar{\mathbf{y}}$ and $\bar{\mathbf{x}}$.

$$\bar{\mathbf{x}} = \sum_{n=1}^{N} \frac{1}{N} x_n \tag{3.55}$$

$$\bar{\mathbf{y}} = \sum_{n=1}^{N} \frac{1}{N} y_n \tag{3.56}$$

$$\tag{3.57}$$

The matrix $\mathbf{B}$ is defined and its singular value decomposition is computed.

$$B = \sum_{n=1}^{N} (\mathbf{y}_n - \bar{\mathbf{y}})(\mathbf{x}_n - \bar{\mathbf{x}})^T \tag{3.58}$$

$$= UDV^T \tag{3.59}$$

The optimal value of $\mathbf{R}^*$ that minimizes Equation 3.54 is:

$$\mathbf{R}^* = \mathbf{V}\mathbf{U}^T \tag{3.60}$$

Since $\mathbf{R}^*$ is an orthonormal matrix that the matrix that minimizes Equation 3.54, it may be either a rotation matrix or a reflection matrix (see Appendix C.1). Therefore the value of the determinant is checked, and if it is equal (or numerically close to) -1, the Equation 3.61.

$$\mathbf{R}^* = \mathbf{V} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \mathbf{U}^T \tag{3.61}$$

Once this is found, the translation is computed using:

$$\mathbf{T}^* = \bar{\mathbf{y}} - \mathbf{R}\bar{\mathbf{x}} \tag{3.62}$$

The source code for this method is found in Appendix B.2.

3.6 Multiplicative Extended Kalman Filter Implementation

In the previous section, photogrammetry has been used to determine a nonlinear least squares estimate of the relative pose between two spacecraft. Although many applications rely on photogrammetry alone, it is useful to filter these estimates with the relative system dynamics. This allows the estimates to be "smoothed" by the dynamics of the system. Additionally, if the target goes out of the view of the camera, the relative states can continue to be propagated without measurement updates. Lastly, the a priori state estimate allows the formulation of outlier hypothesis tests to improve the robustness of the system.

3.6.1 Review of Extended Kalman Filter Equations

The discrete time Extended Kalman Filter (EKF) is reviewed here. Further details can be found in numerous textbooks including [11, 18].

The state vector is represented by $\mathbf{x}(k)$, the measurements are represented by $\mathbf{y}(k)$. The nonlinear dynamic model and measurement equations are given by:

$$\begin{aligned}
\mathbf{x}(k+1) &= \mathbf{f}(\mathbf{x}(k)) + \mathbf{\Gamma W}(k) && (3.63)\\
\mathbf{y}(k) &= \mathbf{h}(\mathbf{x}(k)) + \mathbf{V}(k) && (3.64)\\
E[\mathbf{\Gamma W}(i)\mathbf{W}^T(j)\mathbf{\Gamma}^T] &= \mathbf{Q}\delta(i-j) && (3.65)\\
E[\mathbf{V}(i)\mathbf{V}^T(j)] &= \mathbf{R}\delta(i-j) && (3.66)\\
E[\mathbf{V}(i)\mathbf{W}^T(j)] &= \mathbf{0} && (3.67)\\
& && (3.68)
\end{aligned}$$

The EKF is a minimum mean squared estimate that models the probability distributions as Gaussian distributions within the Hidden Markov Model framework. The a priori estimate (mean of a Gaussian distribution) is given by $\hat{\mathbf{x}}(k)^-$ and the a pos-

terori is given by $\hat{\mathbf{x}}(k)^+$. The a priori covariance matrix for the Gaussian distribution of the state is $\mathbf{P}(k)^-$, while the a posteriori is $\mathbf{P}(k)^+$. The nonlinear dynamics and measurement model is linearized using the a posteriori estimate from the previous iteration.

$$\begin{aligned} \mathbf{\Phi}(k) &= \left.\frac{\partial \mathbf{f}(\mathbf{a})}{\partial \mathbf{x}}\right|_{\mathbf{a}=\hat{\mathbf{x}}(k-1)^+} && (3.69) \\ \mathbf{H}(k) &= \left.\frac{\partial \mathbf{h}(\mathbf{a})}{\partial \mathbf{x}}\right|_{\mathbf{a}=\hat{\mathbf{x}}(k-1)^+} && (3.70) \\ & && (3.71) \end{aligned}$$

The state prediction equations are:

$$\begin{aligned} \hat{\mathbf{x}}(k)^- &= \mathbf{f}(\mathbf{x}(k-1)) && (3.72) \\ \mathbf{P}(k)^- &= \mathbf{\Phi}(k)\mathbf{P}(k-1)^+\mathbf{\Phi}(k)^T + \mathbf{Q} && (3.73) \end{aligned}$$

Now, the measurement update equations are:

$$\begin{aligned} \mathbf{K}(k) &= \mathbf{P}(k)^-\mathbf{H}(k)^T\left(\mathbf{H}(k)\mathbf{P}(k)^-\mathbf{H}(k)^T + \mathbf{R}\right)^{-1} && (3.74) \\ \hat{\mathbf{x}}(k)^+ &= \hat{\mathbf{x}}(k)^- + \mathbf{K}(k)\left(\mathbf{y}(k) - \mathbf{h}(\mathbf{x}(k)^-)\right) && (3.75) \\ \mathbf{P}(k)^+ &= \left(\mathbf{I} - \mathbf{K}(k)\mathbf{H}(k)\right)\mathbf{P}(k)^- && (3.76) \end{aligned}$$

Note that the covariance matrices $\mathbf{P}, \mathbf{Q}$ and $\mathbf{R}$ must always be positive symmetric definite. In order to ensure numerical issues do not allow the state covariance to become non-symmetric, at the end of both the prediction and update equations, the following equation is used: $\mathbf{P} = \frac{1}{2}(\mathbf{P} + \mathbf{P}^T)$. Additionally, Equation 3.77 is used for the measurement update of the state covariance matrix to ensure better numerical stability when $\mathbf{P}(k)^-$ has a large condition number.

$$\mathbf{P}(k)^{+} = (\mathbf{I} - \mathbf{K}(k)\mathbf{H}(k))\,\mathbf{P}(k)^{-}\,(\mathbf{I} - \mathbf{K}(k)\mathbf{H}(k))^{T} + \mathbf{K}(k)\mathbf{R}\mathbf{K}(k)^{T} \quad (3.77)$$

3.6.2 Continuous Time Kinematics and Dynamics of Relative Spacecraft Formations

In the modeling of the system, the forces and torques that are applied by either of the satellites will not be incorporated, and it will be assumed that the angular rates are small. Additionally, gyroscope measurements will not be used. This is done to maintain simplicity and generality, but will likely result in reduced accuracy of the filtering.

The relative position, velocity, orientation and angular velocity is defined by $\mathbf{r}, \mathbf{v}, \mathbf{q}$ and ω respectively. The relative states to be estimated are defined as $\mathbf{x}$.

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} & \mathbf{v} & \mathbf{q} & \omega \end{bmatrix}^{T} \quad (3.78)$$

$$\mathbf{r} = \begin{bmatrix} r_x & r_y & r_z \end{bmatrix}^{T} \quad (3.79)$$

$$\mathbf{v} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^{T} \quad (3.80)$$

$$\mathbf{q} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}^{T} \quad (3.81)$$

$$\omega = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^{T} \quad (3.82)$$

For details on rotational representations see Appendix C.1.

The stochastic continuous time nonlinear dynamics are specified by the following equations. This is a constant linear and angular velocity model with disturbance forces and torques. Various models for tracking are described by [6].

$$\dot{\mathbf{r}} = \mathbf{v} \tag{3.83}$$

$$\dot{\mathbf{v}} = \frac{1}{m}\mathbf{W}_{\mathbf{v}} \tag{3.84}$$

$$\dot{\mathbf{q}} = \frac{1}{2}\boldsymbol{\Omega}(\omega)\mathbf{q} \tag{3.85}$$

$$= \frac{1}{2}\begin{bmatrix}\omega \\ 0\end{bmatrix} \otimes \mathbf{q} \tag{3.86}$$

$$\dot{\omega} = \mathbf{J}^{-1}(-\omega \times \mathbf{J}\omega + \mathbf{W}_{\omega}) \tag{3.87}$$

Given that the relative angular velocity is small, the gyroscopic moment is dropped from the Euler's Rotational Equation, listed in equation 3.87. Therefore:

$$\dot{\omega} \approx \mathbf{J}^{-1}\mathbf{W}_{\omega} \tag{3.88}$$

Where the $\mathbf{J}$ is the inertia matrix, and:

$$\boldsymbol{\Omega}(\omega) = \begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 \\ -\omega_3 & 0 & \omega_1 & \omega_2 \\ \omega_2 & -\omega_1 & 0 & \omega_3 \\ -\omega_1 & -\omega_2 & -\omega_3 & 0 \end{bmatrix} \tag{3.89}$$

Also:

$$[\omega\times] = \begin{bmatrix} 0 & \omega_3 & -\omega_2 \\ -\omega_3 & 0 & \omega_1 \\ \omega_2 & -\omega_1 & 0 \end{bmatrix} \tag{3.90}$$

$\mathbf{W}_{\mathbf{v}}, \mathbf{W}_{\omega}$ are process noise models that incorporate disturbance forces that are applied to *both* spacecraft.

$$E[\mathbf{W_v}(\tau_1)\mathbf{W_v}(\tau_2)^T] = \mathbf{Q_v}\delta(\tau_1 - \tau_2) \tag{3.91}$$

$$E[\mathbf{W}_\omega(\tau_1)\mathbf{W}_\omega(\tau_2)^T] = \mathbf{Q}_\omega\delta(\tau_1 - \tau_2) \tag{3.92}$$

$$E[\mathbf{W_v}(\tau_1)\mathbf{W}_\omega(\tau_2)^T] = \mathbf{0_{3\times 3}} \tag{3.93}$$

The use of a quaternion as a parameterization of the relative attitude is problematic, since a quaternion has four parameters, one constraint equation and three degrees of freedom. This implies that one of the elements of the quaternion is deterministic and not stochastic. As a result, a covariance matrix of a quaternion has one eigenvalue that is exactly zero. Due to small numerical issues, this eigenvalue may become slightly negative, and as a result, the entire state covariance matrix may become non-positive definite, which will result in the divergence of the Extended Kalman Filter. This is discussed in detail in Appendix C.2.

A solution to this problem was fist published by Lefferts in 1982 [59] and further discussed by other authors [18, 65]. This is commonly referred to as the Multiplicative Extended Kalman Filter (MEKF). The concept is to maintain a two parameterizations of the state, one of which is a full reference quaternion $\mathbf{q}_{ref}$ and other is an three element error parameterization $\mathbf{a}_p$ (such as the Modified Rodrigues Parameters, see Appendix C.1.3). The error vector $\mathbf{a}_p$ is used in the propagation and update step, and then a final reset step is introduced where the $\mathbf{q}_{ref}$ is corrected with the a posterior value of $\mathbf{a}_p$, which is then reset to zero for the next iteration. Therefore the actual state vector is:

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} & \mathbf{v} & \mathbf{a}_p & \omega \end{bmatrix}^T \tag{3.94}$$

In order to model the state transitions using $\mathbf{a}_p$, the time derivative $\dot{\mathbf{a}}_p = \frac{d\mathbf{a}_p}{dt}$ must be found. This can be expressed in terms of the elements of $\dot{\mathbf{q}}$ using the quotient rule.

$$\frac{d\mathbf{a}_p}{dt} = \dot{\mathbf{a}}_p = \frac{d}{dt}\left(\frac{4}{1+q_4}\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}\right) \tag{3.95}$$

$$= \frac{4}{(1+q_4)^2}\begin{bmatrix} \dot{q}_1(1+q_4) - \dot{q}_4 q_1 \\ \dot{q}_2(1+q_4) - \dot{q}_4 q_2 \\ \dot{q}_3(1+q_4) - \dot{q}_4 q_3 \end{bmatrix} \tag{3.96}$$

3.6.3 Discrete Time Kinematics and Dynamics of Relative Spacecraft Formations

In order to discretize the system, it must first be linearized. Note that the linearization point for $\mathbf{a}_p = \mathbf{0}_{3\times 1}$.

$$\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{B}_W\mathbf{W} \tag{3.97}$$

$$\begin{bmatrix} \dot{\mathbf{r}} \\ \dot{\mathbf{v}} \\ \dot{\mathbf{a}}_p \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \frac{1}{2}[\omega\times] & \mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \mathbf{v} \\ \mathbf{a}_p \\ \omega \end{bmatrix} \tag{3.98}$$

$$+ \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \frac{1}{m}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{J}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{W}_{\mathbf{v}} \\ \mathbf{W}_{\omega} \end{bmatrix} \tag{3.99}$$

The discrete time equations are derived for a time-step Δt of the continuous model are given by the following equations. Notice that it is assumed that the process noise is piecewise constant during the time-step (zero order hold).

$$\mathbf{x}(k) = \mathbf{e}^{\mathbf{A}\Delta t}\mathbf{x}(k-1) + \int_0^{\Delta T} \mathbf{e}^{\mathbf{A}\tau}\mathbf{B}_W\mathbf{W}d\tau \tag{3.100}$$

$$= \mathbf{e}^{\mathbf{A}\Delta t}\mathbf{x}(k-1) + \left(\int_0^{\Delta T} \mathbf{e}^{\mathbf{A}\tau}\mathbf{B}_W d\tau\right)\mathbf{W}(k) \tag{3.101}$$

$$= \mathbf{\Phi}(k)\mathbf{x}(k-1) + \mathbf{\Gamma}(k)\mathbf{W}(k) \tag{3.102}$$

$$\tag{3.103}$$

Now, the values of these matricies are:

$$\mathbf{\Phi}(k) = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{I}_{3\times3}\Delta t & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{e}^{\frac{1}{2}[\omega\times]\Delta t} & \int_0^{\Delta t}\mathbf{e}^{\frac{1}{2}[\omega\times]\tau}d\tau \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix} \tag{3.104}$$

$$\tag{3.105}$$

$$\mathbf{\Gamma}(k) = \begin{bmatrix} \frac{1}{2m}\mathbf{I}_{3\times3}\Delta t^2 & \mathbf{0}_{3\times3} \\ \frac{1}{m}\mathbf{I}_{3\times3}\Delta t & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \Delta t\int_0^{\Delta t}\mathbf{e}^{\frac{1}{2}[\omega\times]\tau}\mathbf{J}^{-1}d\tau \\ \mathbf{0}_{3\times3} & \mathbf{J}^{-1}\Delta t \end{bmatrix} \tag{3.106}$$

Since the analytical solutions to the above integrals are complicated, and numerically unstable at very small angular rates, these equations will be solved numerically using the matrix exponential at each iteration. Two sub-matricies of the linearized dynamics are defined as $\mathbf{A}_a, \mathbf{B}_a$.

$$\begin{bmatrix} \dot{\mathbf{a}}_p \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}[\omega\times] & \mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \begin{bmatrix} \mathbf{a}_p \\ \omega \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{3\times3} \\ \mathbf{J}^{-1} \end{bmatrix} \mathbf{W}_\omega \tag{3.107}$$

$$= \mathbf{A}_a \begin{bmatrix} \mathbf{a}_p \\ \omega \end{bmatrix} + \mathbf{B}_a \mathbf{W}_\omega \tag{3.108}$$

Creating a temporary system, that is augmented with the process noise, $\mathbf{A}_M$ is defined:

$$\begin{bmatrix} \dot{\mathbf{a}}_p \\ \dot{\omega} \\ \dot{\mathbf{W}}_\omega \end{bmatrix} = \begin{bmatrix} \frac{1}{2}[\omega\times] & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{J}^{-1} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \begin{bmatrix} \mathbf{a}_p \\ \omega \\ \mathbf{W}_\omega \end{bmatrix} \tag{3.109}$$

$$= \mathbf{A}_M \begin{bmatrix} \mathbf{a}_p \\ \omega \\ \mathbf{W}_\omega \end{bmatrix} \tag{3.110}$$

$$\tag{3.111}$$

Now during each iteration, $\mathbf{\Phi}_M$ is solved for using numerical methods to compute the matrix exponential.

$$\mathbf{\Phi}_M = e^{\mathbf{A}_M \Delta t} \tag{3.112}$$

$$= \begin{bmatrix} \mathbf{\Phi}_{M11} & \mathbf{\Phi}_{M12} & \mathbf{\Phi}_{M13} \\ \mathbf{\Phi}_{M21} & \mathbf{\Phi}_{M22} & \mathbf{\Phi}_{M23} \\ \mathbf{\Phi}_{M31} & \mathbf{\Phi}_{M32} & \mathbf{\Phi}_{M33} \end{bmatrix} \tag{3.113}$$

Now the following relationships can be found that will solve for the block elements of Equations 3.104 and 3.106 that require numerical integration.

$$\int_0^{\Delta t} \mathbf{e}^{\frac{1}{2}[\omega\times]\tau} d\tau = \mathbf{\Phi}_{M12} \tag{3.114}$$

$$\Delta t \int_0^{\Delta t} \mathbf{e}^{\frac{1}{2}[\omega\times]\tau} \mathbf{J}^{-1} d\tau = \mathbf{\Phi}_{M13} \tag{3.115}$$

As a check, the values of other block elements should be identical to what was previously found:

$$\mathbf{\Phi}_{M11} = \mathbf{e}^{\frac{1}{2}[\omega\times]\Delta t} \tag{3.116}$$

$$\mathbf{\Phi}_{M21} = \mathbf{0}_{3\times3} \tag{3.117}$$

$$\mathbf{\Phi}_{M22} = \mathbf{I}_{3\times3} \tag{3.118}$$

$$\mathbf{\Phi}_{M23} = \mathbf{J}^{-1}\Delta t \tag{3.119}$$

$$\mathbf{\Phi}_{M31} = \mathbf{0}_{3\times3} \tag{3.120}$$

$$\mathbf{\Phi}_{M32} = \mathbf{0}_{3\times3} \tag{3.121}$$

$$\mathbf{\Phi}_{M33} = \mathbf{I}_{3\times3} \tag{3.122}$$

As a separate check, in the case where the angular velocities are zero:

$$\mathbf{\Phi}_{M12} = \mathbf{I}_{3\times3}\Delta t \tag{3.123}$$

$$\mathbf{\Phi}_{M13} = \frac{1}{2}\mathbf{J}^{-1}\Delta t^2 \tag{3.124}$$

These source code in Appendix B.3 was verified using these checks.

3.6.4 Measurement Update Equations

Using the output of the exterior orientation algorithm described in Section 3.5 as measurements $\check{\mathbf{r}}(k)$ and $\check{\mathbf{q}}(k)$ are used as inputs to the MEKF measurement update equations as follows. The first step is to determine the measurement of the attitude

error $\check{\mathbf{a}}_p(k)$. Note that $\mathbf{q}_{ref}(k)^*$ is the inverse of the reference rotation.

$$\delta\mathbf{q}(\check{\mathbf{a}}_p(k)) = \check{\mathbf{q}}(k) \otimes \mathbf{q}_{ref}(k)^* \tag{3.125}$$

$$\mathbf{y}(k) = \begin{bmatrix} \check{\mathbf{r}}(k) \\ \check{\mathbf{a}}_p(k) \end{bmatrix} = \mathbf{H}(k)\mathbf{x}(k) + \mathbf{V} \tag{3.126}$$

$$= \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \begin{bmatrix} \mathbf{r}(k) \\ \mathbf{v}(k) \\ \mathbf{a}_p(k) \\ \omega(k) \end{bmatrix} \tag{3.127}$$

$$+ \begin{bmatrix} \mathbf{V}_r \\ \mathbf{V}_{a_p} \end{bmatrix} \tag{3.128}$$

Equation 3.127 shows that the measurement model is linear given the solution to the exterior orientation problem.

The values of $\mathbf{\Phi}(k), \mathbf{\Gamma}(k), \mathbf{H}(k)$ and $\mathbf{f}(\mathbf{x}_k)$ are used to run the MEKF and determine values of $\hat{\mathbf{x}}(k)^-, \hat{\mathbf{x}}(k)^+, \mathbf{P}(k)^-, \mathbf{P}(k)^+$ at each iteration based on measurements $\mathbf{y}(k)$ from the solution to the exterior orientation problem.

3.6.5 Reset Step

At the end of each iteration, the reset step applies the following equations:

$$\hat{\mathbf{q}}(k) = \delta\mathbf{q}(\hat{\mathbf{a}}_p^+(k)) \otimes \mathbf{q}_{ref}(k) \tag{3.129}$$

$$\hat{\mathbf{a}}_p^+(k) = \mathbf{0}_{3\times1} \tag{3.130}$$

$$\mathbf{q}_{ref}(k+1) = \hat{\mathbf{q}}(k) \tag{3.131}$$

3.6.6 Fault Detection and Outlier Rejection

In this approach, there is the possibility that the image processing algorithms may incorrectly detect another object in the scene as a visual marker. Normally, the algorithm will only proceed to the exterior orientation stage if there is exactly four targets detected. However, in some situations there may be exactly four markers detected, but one or more of them are false positives (for example if one false positive and one false negative occur simultaneously).

In this situation it is unlikely that the result of the exterior orientation will be close to the correct location. In other words, the probability distribution of the relative location and orientation for this type of erroneous measurement is uniform over the field of view of the camera. This is commonly referred to as outlier rejection, and is often handled in a Kalman Filter by "innovation filtering". In the SPHERES estimator, the L1 norm of the innovation is checked to see whether it as above or below a preset threshold. If it is above the threshold, the measurement is thrown out [75].

One problem with this approach is that it does not consider, how accurate the current estimate is. For example, if the system experiences an un-modeled disturbance, many of the measurements will have high innovations; if these innovations are above the threshold (and therefore rejected) the filter will diverge.

An outlier rejection method that takes into account the accuracy of the current estimate is desirable. For this purpose the square of the Mahalanobis distance of the innovation was considered [88]. The innovation is $\mathbf{i}(k) = \mathbf{y}(k) - \mathbf{h}(\hat{\mathbf{x}}(k))$. The Mahalanobis distance is defined as $\mathbf{d}(k)$:

$$\mathbf{d}(k) \quad = \quad \sqrt{\mathbf{i}(k)^T \left(E[\mathbf{i}(k)\mathbf{i}(k)^T]\right)^{-1} \mathbf{i}(k)} \tag{3.132}$$

Where

$$
\begin{aligned}
E[\mathbf{i}(k)\mathbf{i}(k)^T] &= E[(\mathbf{y}(k) - \mathbf{h}(\hat{\mathbf{x}}(k)^-))(\mathbf{y}(k) - \mathbf{h}(\hat{\mathbf{x}}(k)^-))^T] && (3.133)\\
&= E[\mathbf{y}(k)\mathbf{y}(k)^T] - 2E[\mathbf{y}(k)\mathbf{h}(\hat{\mathbf{x}}(k)^-)^T] && (3.134)\\
&\quad + E[\mathbf{h}(\hat{\mathbf{x}}(k)^-)\mathbf{h}(\hat{\mathbf{x}}(k)^-)^T] && (3.135)\\
&= E[\mathbf{y}(k)\mathbf{y}(k)^T] - 2E[\mathbf{y}(k)]E[\mathbf{h}(\hat{\mathbf{x}}(k)^-)^T] && (3.136)\\
&\quad + E[\mathbf{h}(\hat{\mathbf{x}}(k)^-)\mathbf{h}(\hat{\mathbf{x}}(k)^-)^T] && (3.137)\\
&= E[\mathbf{y}(k)\mathbf{y}(k)^T] + E[\mathbf{h}(\hat{\mathbf{x}}(k)^-)\mathbf{h}(\hat{\mathbf{x}}(k)^-)^T] && (3.138)\\
&\approx E[\mathbf{y}(k)\mathbf{y}(k)^T] + \mathbf{H}(k)E[\hat{\mathbf{x}}(k)^-\hat{\mathbf{x}}(k)^{-T}]\mathbf{H}(k)^T && (3.139)\\
&= \mathbf{R} + \mathbf{H}(k)\mathbf{P}(k)^-\mathbf{H}(k)^T && (3.140)
\end{aligned}
$$

Using the above covariance, the value of the square of the Mahalanobis distance is checked against a threshold T.

$$
\begin{aligned}
T &> \mathbf{d}(k)^2 && (3.141)\\
T &> (\mathbf{y}(k) - \mathbf{h}(\hat{\mathbf{x}}(k)))^T \left(\mathbf{R} + \mathbf{H}(k)\mathbf{P}(k)^-\mathbf{H}(k)^T\right)^{-1}\\
&\qquad \times(\mathbf{y}(k) - \mathbf{h}(\hat{\mathbf{x}}(k))) && (3.142)
\end{aligned}
$$

In this application, it was found to be useful to tune two separate thresholds for the position and orientation measurements and to exclude the linear and angular velocities.

$$
\begin{aligned}
T_\mathbf{r} &> (\mathbf{y}_\mathbf{r}(k) - \mathbf{h}_\mathbf{r}(\hat{\mathbf{x}}(k)))^T \left(\mathbf{R}_\mathbf{r} + \mathbf{H}_\mathbf{r}(k)\mathbf{P}_\mathbf{r}(k)^-\mathbf{H}_\mathbf{r}(k)^T\right)^{-1}\\
&\qquad \times (\mathbf{y}_\mathbf{r}(k) - \mathbf{h}_\mathbf{r}(\hat{\mathbf{x}}(k))) && (3.143)\\
T_\mathbf{q} &> (\mathbf{y}_\mathbf{q}(k) - \mathbf{h}_\mathbf{q}(\hat{\mathbf{x}}(k)))^T \left(\mathbf{R}_\mathbf{q} + \mathbf{H}_\mathbf{q}(k)\mathbf{P}_\mathbf{q}(k)^-\mathbf{H}_\mathbf{q}(k)^T\right)^{-1}\\
&\qquad \times(\mathbf{y}_\mathbf{q}(k) - \mathbf{h}_\mathbf{q}(\hat{\mathbf{x}}(k))) && (3.144)
\end{aligned}
$$

3.6.7 Implementation and Tuned Parameters

An implementation of the Multiplicative Extended Kalman Filter requires the selection of error covariances for the process and measurement noise ($\mathbf{W}$ in Equation 3.97 and $\mathbf{V}$ in Equation 3.126) in order to tune the filter gains.

One important point is that the measurement noise should be proportional to the re-projection error ϵ_k^2 from Equation 3.49. Although a physically based initial estimate of the covariances was tried, extensive tuning led to the following parameters for the process and measurement noise covariance matrices (at a given instant in time):

$$E[\mathbf{W}\mathbf{W}^T] = \begin{bmatrix} 4.0 \times 10^{-4}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & 4.7873 \times 10^{-5}\mathbf{I}_{3\times3} \end{bmatrix} \tag{3.145}$$

$$E[\mathbf{V}\mathbf{V}^T] = \begin{bmatrix} \epsilon^2 \times 10^{-5}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \epsilon^2 \times 1.5625\mathbf{I}_{3\times3} \end{bmatrix} \tag{3.146}$$

$$\tag{3.147}$$

These parameters are tuned for an update rate of 3 Hz. The stopping condition for the exterior orientation algorithm leads to $\epsilon^2 = 5 \times 10^{-5}$, assuming the algorithm is not stopped prematurely due to computational time constraints.

The threshold for the outlier rejection method in Equation 3.142 was tuned experimentally, through trial and error by adjusting experimentally gathered data. Both $T_{\mathbf{r}}$ and $T_{\mathbf{q}}$ were set to a value of 30,000. Figure 3-16 shows the Mahalanobis distances (multiplied by -1) for a series of experimental data, with a 10 cm incorrect measurement manually inserted at 20.77 seconds. It clearly shows that there is a very strong peak that is greater than 30,000. Because the Mahalanobis distance is also a Chi-Squared distribution with a zero mean, the probability of a false negative (i.e. a correct measurement incorrectly being thrown out as an outlier) can be found by computing the area under the curve of a Chi-Squared Distribution between the $q = T$ and infinity. When this was computed in Matlab, the result is zero, which implies the probability is less than the floating point precision of matlab (2.2×10^{-16} in this

case). This strongly indicates that it is unlikely to get a false positive, and implies that the threshold could be lowered to catch even more outliers.

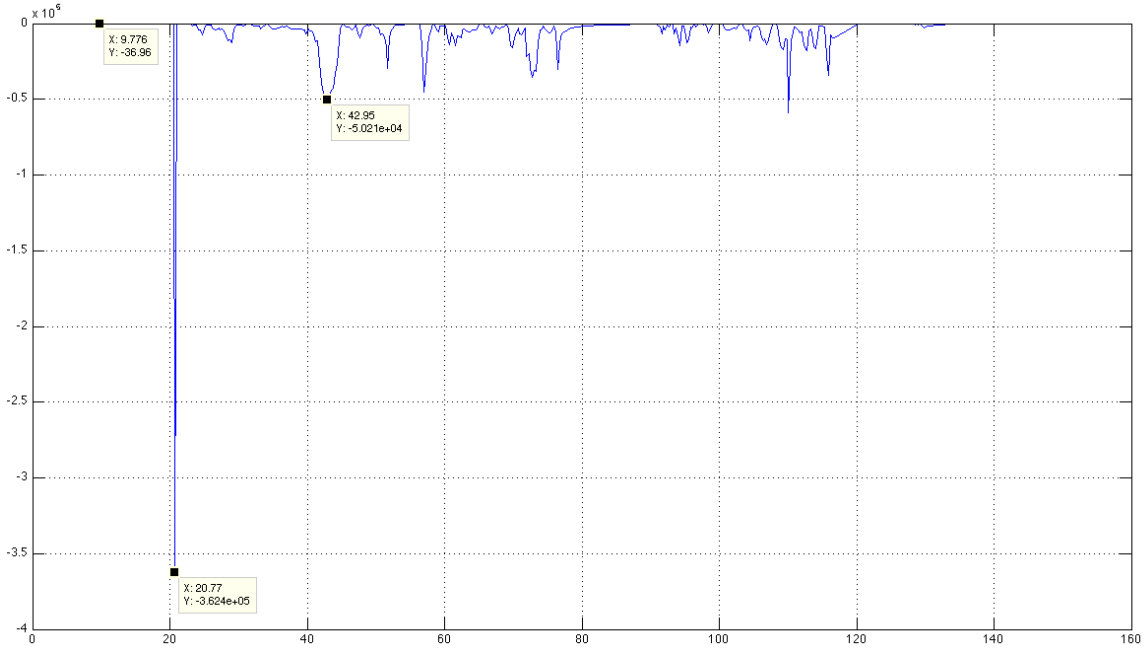


Figure 3-16: Plot of Mahalanobis Distance with 10 cm Outlier at $t = 20.77$ Seconds

This overall approach was implemented in C on the SPHERES Goggles. Significant portions of the code were developed in Matlab and then converted to C using the Matlab Real-Time Workshop. The Matlab code can be found in Appendix B.

3.7 Experimental Characterization of Accuracy and Precision

In order to characterize the accuracy and precision of this approach, the image processing accuracy is first investigated in Section 3.7.1. The next step is to investigate the error of the exterior orientation, which is presented in Section 3.7.2 and is followed by the validation of the Multiplicative Extended Kalman Filter in Section 3.7.3.

3.7.1 Image Processing

The first question with regards to the precision of the Concentric Circle Tracker discussed in Section 3.4 is to determine whether or not the precision is limited by diffraction, or by the size of an individual pixel.

The characteristics of the camera and lens are listed in Table 3.2. Notice that the relatively low F-Stop number indicates that there is a poor depth of field.

Table 3.2: Optics Characteristics

Pixel Size	6.0 μ m
Focal Length	630 pixels or 3.78 mm
Lens Diameter	8.05 mm
Peak Wavelenght	700 nm
F-Stop Number	0.47

Using this information, the Rayleigh Diffraction limit can be found as follows:

$$\Theta_R = \frac{1.22\lambda}{D} \tag{3.148}$$

$$= 0.006^o \tag{3.149}$$

The angle that is subtended by a single pixel ranges from 0.07^o at the focal point, to 0.09^o at the edge of the image. This indicates that the accuracy is limited by the size of the pixels.

If an object is found to be located within a single pixel, it means that it is found with a precision of approximately 0.8^o. This implies that if this object is a distance of 1 meter from the camera, the location of the object would be known to within 1.4 millimeters, with uniform probability as to where within this pixel it is located.

The next step to characterizing the image processing algorithm was to determine what type of noise is present in the location of the CCC target's centroid. By keeping the location of the camera and target static, it was found that the centroid location typically varied around 0.2 pixels from frame to frame. This led to the guess that the centroid noise could likely be modeled by a zero mean white gaussian noise in two dimensions (x and y) with a standard deviation of approximately 0.2 pixels.

3.7.2 Exterior Orientation

Distance Measurement Comparison using a Tape Measure

The Exterior Orientation algorithm (described in Section 3.5) outputs the relative position and orientation $\mathbf{r}_{T/C}$ and $\mathbf{R}_{TC}$ described in Equations 3.2 and 3.2. The first step in characterizing the errors in the output of this algorithm is to measure the distance $||\mathbf{r}_{T/C}||_2$ and compare it to what can be measured by hand using a tape measure and a length of string. The results of a number of different positions are shown in Table 3.3. It is estimated that the precision of the tape measure distances is ± 0.1 cm, and the precision of the exterior orientation solution is ± 0.1 cm, based strictly on repeated trials.

Table 3.3: Comparison of Straight Line Distance to Target between Exterior Orientation and Tape Measure

Number	**Exterior Orientation:** $\mathbf{r}_{T/C} = \begin{bmatrix} x & y & z \end{bmatrix}$	**Exterior Orientation:** $\|\|\mathbf{r}_{T/C}\|\|_2$	**Tape Measure Distance:** $\|\|\mathbf{r}_{T/C}\|\|_2$	**Difference**
1	$\begin{bmatrix} -10.6 & 1.9 & 40.4 \end{bmatrix}$ cm	41.81 cm ± 0.1 cm	41.3 cm ± 0.1 cm	0.51 cm ± 0.2 cm
2	$\begin{bmatrix} -3.7 & 1.7 & 31.9 \end{bmatrix}$ cm	32.15 cm ± 0.1 cm	32.7 cm ± 0.1 cm	0.55 cm ± 0.2 cm
3	$\begin{bmatrix} -10.2 & 2.5 & 90.7 \end{bmatrix}$ cm	91.31 cm ± 0.1 cm	91.4 cm ± 0.1 cm	0.09 cm ± 0.2 cm
4	$\begin{bmatrix} 15.1 & 1.6 & 45.6 \end{bmatrix}$ cm	48.06 cm ± 0.1 cm	48.1 cm ± 0.1 cm	0.04 cm ± 0.2 cm
5	$\begin{bmatrix} -19.0 & 2.1 & 45.6 \end{bmatrix}$ cm	49.44 cm ± 0.1 cm	49.5 cm ± 0.1 cm	0.06 cm ± 0.2 cm

The data in Table 3.3 shows that the straight line distance accuracy of the exterior orientation algorithm varies between 0.5 cm and 0.1 cm. This appears to be correlated

to the location of the target in the image. When the target is close to the center of the image, its accuracy is smaller then when the target is close to the edge of the image. This can be explained by imperfect estimation of the lens distortion parameters as described in Section 3.3.2.

Relative Position and Orientation

The next step in the process of characterizing the accuracy and precision is to compare the estimates of the relative position and orientation as the SPHERES move over time between known positions and orientations. This test will characterize static estimates, by allowing the SPHERE to stop moving and the vision system to take multiple measurements over a period of at least 20 seconds (measurements were taken at 3 Hz). The difficulty in this comparison is ensuring the SPHERES are placed as precisely as possible at the known positions and orientations, as well as determining an estimate of what the error in this placement is.

For this purpose, the SPHERES air carriages were used on a "frictionless and flat" table. The table is known to have some imperfections, but since this test will be a static test, this is not a problem. The table has a visible 5 cm $\times$ 5 cm grid underneath it (see Figure 3-1), which was used along with tick-marks on the air carriage to assist with manual positioning of the SPHERES. It is estimated that the SPHERES were placed with an accuracy of approximately 3 mm and 2 degrees in the x and y directions.

Since this test measures the relative position and orientation of the SPHERES, it utilizes the coordinate transformations described by Equations 3.2 and 3.3, and validates the model parameters specified in Equations 3.4, 3.5, 3.6 and 3.7.

The first experiment analyzes relative orientation data. The sequence of rotations is described in Table 3.4, which shows the schedule of orientations for the data in Figure 3-17.

Table 3.4: Relative Orientation Schedule

Time	**Z-Axis Rotation**	**Y-Axis Rotation**	**X-Axis Rotation**
0 s	-90.0 ± 2.0 deg	0.0 ± 2.0 deg	0.0 ± 2.0 deg
30 s	-45.0 ± 2.0 deg	0.0 ± 2.0 deg	0.0 ± 2.0 deg
60 s	0.0 ± 2.0 deg	0.0 ± 2.0 deg	0.0 ± 2.0 deg
100 s	-45.0 ± 2.0 deg	0.0 ± 2.0 deg	0.0 ± 2.0 deg
120 s	-90.0 ± 2.0 deg	0.0 ± 2.0 deg	0.0 ± 2.0 deg

Since it is likely that there was an initial offset, the changes in orientation over time will be compared first. In Figure 3-17, the Z-Axis rotation sequence is:

$$\theta_1 = -48.53^o - (-94.71^o) = 46.18^o \tag{3.150}$$

$$\theta_2 = -1.14^o - (-48.53^o) = 47.39^o \tag{3.151}$$

$$\theta_3 = -48.98^o - (-1.14^o) = -47.84^o \tag{3.152}$$

$$\theta_4 = -94.79^o - (-48.98^o) = -45.81^o \tag{3.153}$$

$$\tag{3.154}$$

Comparing θ_1 and θ_4 as well as θ_2 and θ_3, it is evident that the rotations are repeatable to within 0.5^o. However, there appears to be a bias (when compared to Table 3.4) in the rotations that ranges from 1.18^o to 2.84^o. This could either be due to the fact that the rotations were not exactly 45^o due to human error in the tickmarks and manual rotation, or this could be due to an intrinsic bias in the measurements. Therefore we can conclude that the error has an upper bound of approximately 3.0^o and a lower bound of 0.5^o.

The Y-Rotation data in Figure 3-17 shows that there is a 1^o bias that is constant throughout the test. This may be due to the fact that the SPHERE is not perfectly vertical in the air carriage, or that the there is a 1^o tilt somewhere in the coordinate transformations that is not modeled by Equations 3.6 and 3.7. Additionally, there are sharp spikes (up to 5^o) in the Y-Rotation angle. It is believed that this is due to the SPHERE shifting in the air carriage as it is being moved, since the spikes in the Y-axis rotation typically occurs about 1 second prior to the SPHERE beginning to

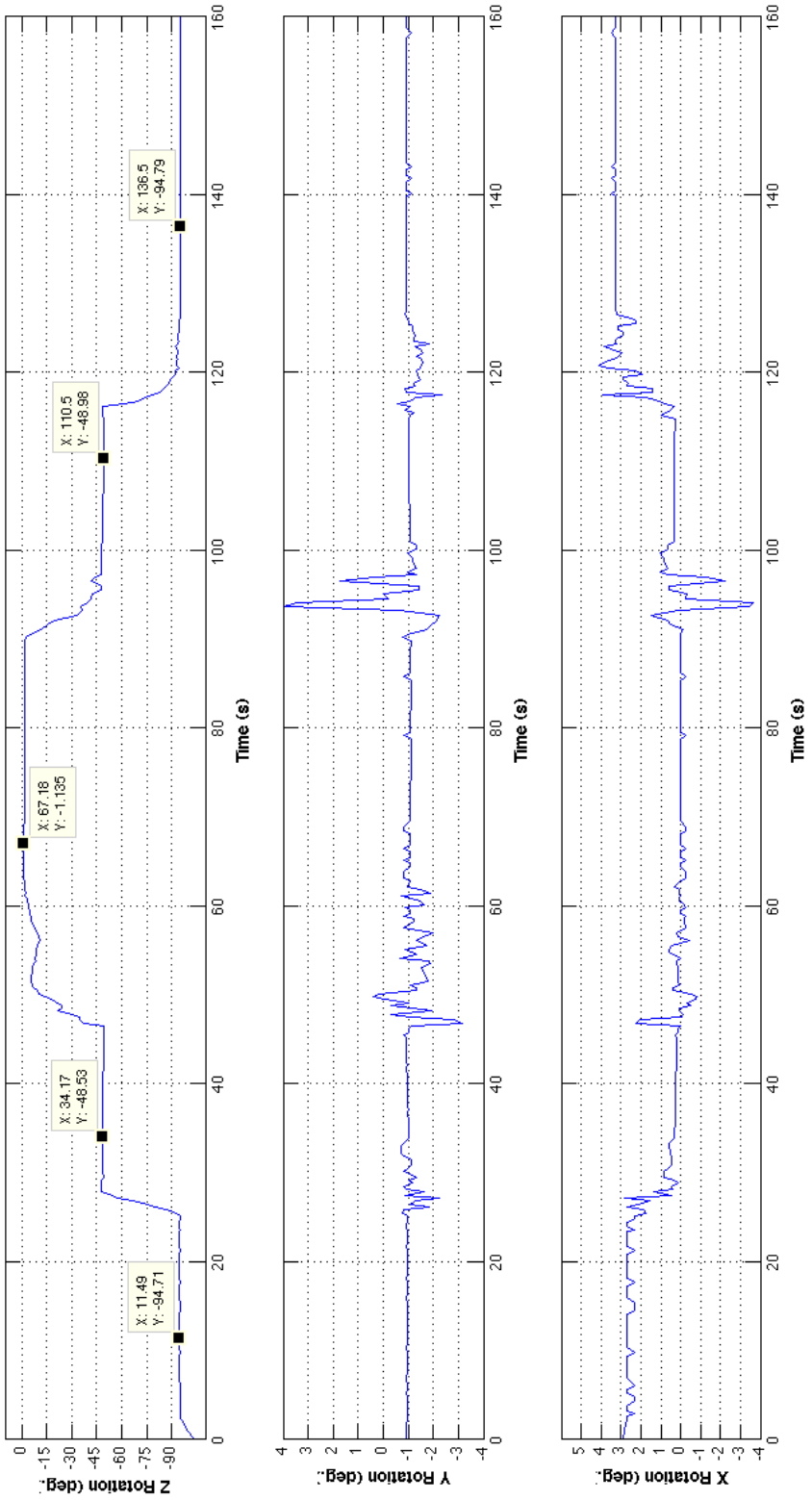


Figure 3-17: Relative Orientation Data From Exterior Orientation

rotate around the Z-axis, which indicates that this is not an anomaly in the exterior orientation algorithm, and in fact the actual movement of the SPHERE.

The X-Rotation data in Figure 3-17 shows a bias that changes over time. Between approximately 30 seconds and 100 seconds, the SPHERES relative rotation about the X-axis is approximately zero. However, when the relative orientation is approximately -90^o there appears to be a positive 2.7^o bias between 0 and 30 seconds and a positive bias of 3.3^o. Since it is highly unlikely that the SPHERE happened to always shift to the same rotation about the x-axis whenever the z-axis rotation is 90^o, it is likely that this is due to the fact that there is a bias in the exterior orientation algorithm. It was discussed in Section 3.5, the exterior orientation algorithm is guaranteed to produce a biased solution if it is limited to a finite number of iterations (however it is globally convergent over an infinite number of iterations). It is conceivable that this biased solution might be a few degrees off of the correct solution, but still falls within the mean-squared re-projection error threshold. It is believed that the x-axis rotation data in Figure 3-17 is an example of this.

It is important to note that a 45^o rotation off of the optical axis is near the practical limit of the system. The image processing algorithm, will have a higher uncertainty for the centroid of each concentric contrasting circle due to the fact that more surface area of the target is compressed into each pixel. An image taken from the camera that is detecting a target at a 45^o rotation off of the optical axis is shown in Figure 3-18 (X's are overlaid on each target, and a box is drawn using the estimate relative position and orientation that surrounds the target border).

From the above analysis, it is concluded that the orientation data has an accuracy (bias) of 3.0^o and a precision (repeatability) of 0.5^o.

The next step was to analyze both the position and orientation solutions of the exterior orientation algorithm. The sequence of points that were travelled to and their estimated "manual positioning" errors are shown in Table 3.5. These points have been rotated by 45 degrees to make it easier to visualize. Notice that after 120 seconds, the locations are repeated in the reverse order. The experimental data collected from the exterior orientation algorithm is shown in Figure 3-19 and 3-20.

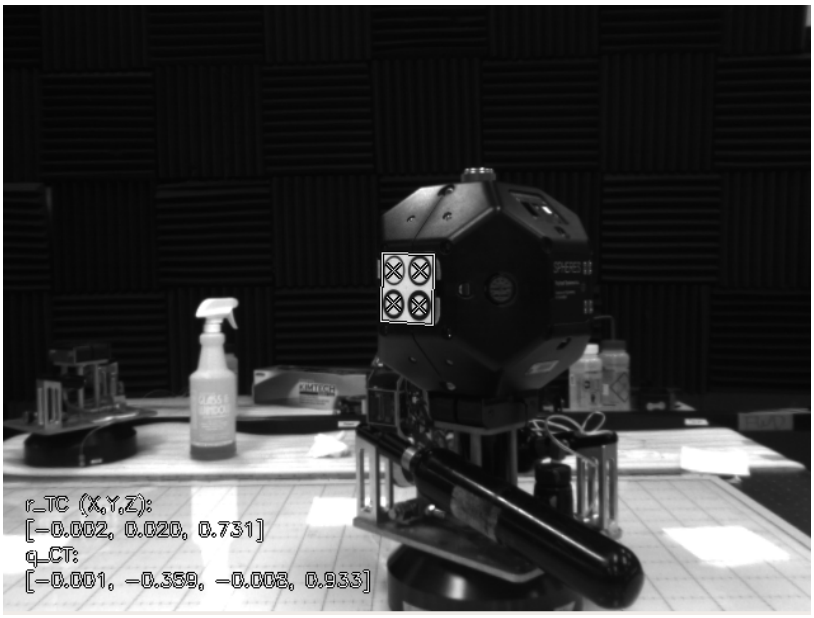


Figure 3-18: Image taken of Target at 45^o off the Optical Axis

Table 3.5: Relative Position and Orientation Schedule

Time	**Relative X-Position**	**Relative Y-Position**	**Relative Z-Position**	**Z-Axis Rotation**	**Y-Axis Rotation**	**X-Axis Rotation**
0 s	60.0 ± 0.3 cm	0.0 ± 0.3 cm	0.0 ± 2.0 cm	0.0 ± 2.0 deg	0.0 ± 2.0 deg	0.0 ± 2.0 deg
40 s	110.0 ± 0.3 cm	25.0 ± 0.3 cm	0.0 ± 2.0 cm	0.0 ± 2.0 deg	0.0 ± 2.0 deg	0.0 ± 2.0 deg
80 s	110.0 ± 0.3 cm	-30.0 ± 0.3 cm	0.0 ± 2.0 cm	0.0 ± 2.0 deg	0.0 ± 2.0 deg	0.0 ± 2.0 deg
120 s	110.0 ± 0.3 cm	0.0 ± 0.3 cm	0.0 ± 2.0 cm	0.0 ± 2.0 deg	0.0 ± 2.0 deg	0.0 ± 2.0 deg
205 s	110.0 ± 0.3 cm	25.0 ± 0.3 cm	0.0 ± 2.0 cm	0.0 ± 2.0 deg	0.0 ± 2.0 deg	0.0 ± 2.0 deg
250 s	110.0 ± 0.3 cm	-30.0 ± 0.3 cm	0.0 ± 2.0 cm	0.0 ± 2.0 deg	0.0 ± 2.0 deg	0.0 ± 2.0 deg
280 s	60.0 ± 0.3 cm	0.0 ± 0.3 cm	0.0 ± 2.0 cm	0.0 ± 2.0 deg	0.0 ± 2.0 deg	0.0 ± 2.0 deg

The data shown for the changes in distance for the X and Y axis can be summarized as follows:

$$\begin{bmatrix} \Delta X_1 \\ \Delta Y_1 \end{bmatrix} = \begin{bmatrix} 110.1 - 59.96 \\ 23.44 - 1.01 \end{bmatrix} = \begin{bmatrix} 50.14 \\ 22.43 \end{bmatrix} \text{cm} \tag{3.155}$$

$$\begin{bmatrix}\Delta X_2\\ \Delta Y_2\end{bmatrix} = \begin{bmatrix}110.6-110.1\\ (-29.82)-23.44\end{bmatrix} = \begin{bmatrix}0.5\\ -53.26\end{bmatrix}\text{ cm} \tag{3.156}$$

$$\begin{bmatrix}\Delta X_3\\ \Delta Y_3\end{bmatrix} = \begin{bmatrix}110.5-110.6\\ -1.08-(-29.82)\end{bmatrix} = \begin{bmatrix}0.1\\ 28.74\end{bmatrix}\text{ cm} \tag{3.157}$$

$$\begin{bmatrix}\Delta X_4\\ \Delta Y_4\end{bmatrix} = \begin{bmatrix}111.4-110.5\\ 24.52-(-1.08)\end{bmatrix} = \begin{bmatrix}0.9\\ 25.60\end{bmatrix}\text{ cm} \tag{3.158}$$

$$\begin{bmatrix}\Delta X_5\\ \Delta Y_5\end{bmatrix} = \begin{bmatrix}111.4-111.4\\ -30.61-24.52\end{bmatrix} = \begin{bmatrix}0.0\\ 55.13\end{bmatrix}\text{ cm} \tag{3.159}$$

$$\begin{bmatrix}\Delta X_6\\ \Delta Y_6\end{bmatrix} = \begin{bmatrix}59.79-111.4\\ 0.01-(-30.61)\end{bmatrix} = \begin{bmatrix}-51.61\\ 30.62\end{bmatrix}\text{ cm} \tag{3.160}$$

Looking at the above changes in location and the raw data in Figure 3-19, it is clear that the x-axis is accurate to less than 3 mm at close range (60 cm), and was accurate to less than 1.5 cm at a longer range (110 cm). The repeatability (precision) of the x-axis data was less than 5 mm. The Y-axis relative position shows that there can be a bias of approximately 1.0 cm, and a repeatability (precision) of less than 5 mm. The Z-axis data stays within a 1.0 cm range, except for the period of time between 120 s and 170 s, where it is 2.0 cm. It is useful to note that the attitude data for the same test (shown in Figure 3-20) has an error in the X and Y rotation axes that is directly correlated with the error in the 120 to 170 second timeframe. It is believed that this is another example of poor convergence of attitude data, possibly while combined with an actual attitude shift of the SPHERES satellites' relative orientation. This is further justified by the fact that in Equation 3.4, $\mathbf{R}_{TC}$ is multiplied by $\mathbf{r}_{2/T}$, which

has a length along the z-axis of 11.3 cm. If this distance is rotated by a 5 degree error, it will also look like a 1 cm difference in the Z-axis ($11.3 \times \sin(5^o) = 0.98$), adding this to the already existing 1.0 cm bias due to possible mis-positioning, and this gives the 2.0 cm bias that is observed.

In order to further investigate the convergence characteristics of the exterior orientation algorithm, additional data was taken for a static relative position and orientation. Within a few time-steps of the algorithm (approximately 1-2 seconds), the exterior orientation will converge to the specified re-projection threshold. However it may still be stuck in a local minimum. When this occurs, only two iterations of the exterior orientation algorithm are run at each time step. The results of this are shown in Figure 3-21. This figure shows a very slow shift in some of the axes. In Figure 3-22, the exterior orientation algorithm was forced to always run 50 iterations of the algorithm per time-step even if the re-projection error was below the threshold. This data shows that an initial value was converged to, however after approximately 25 seconds, there was a shift in the value of the position and orientation. This shows that there is a bias of approximately 2 cm and 3^o, which is significant for this system.

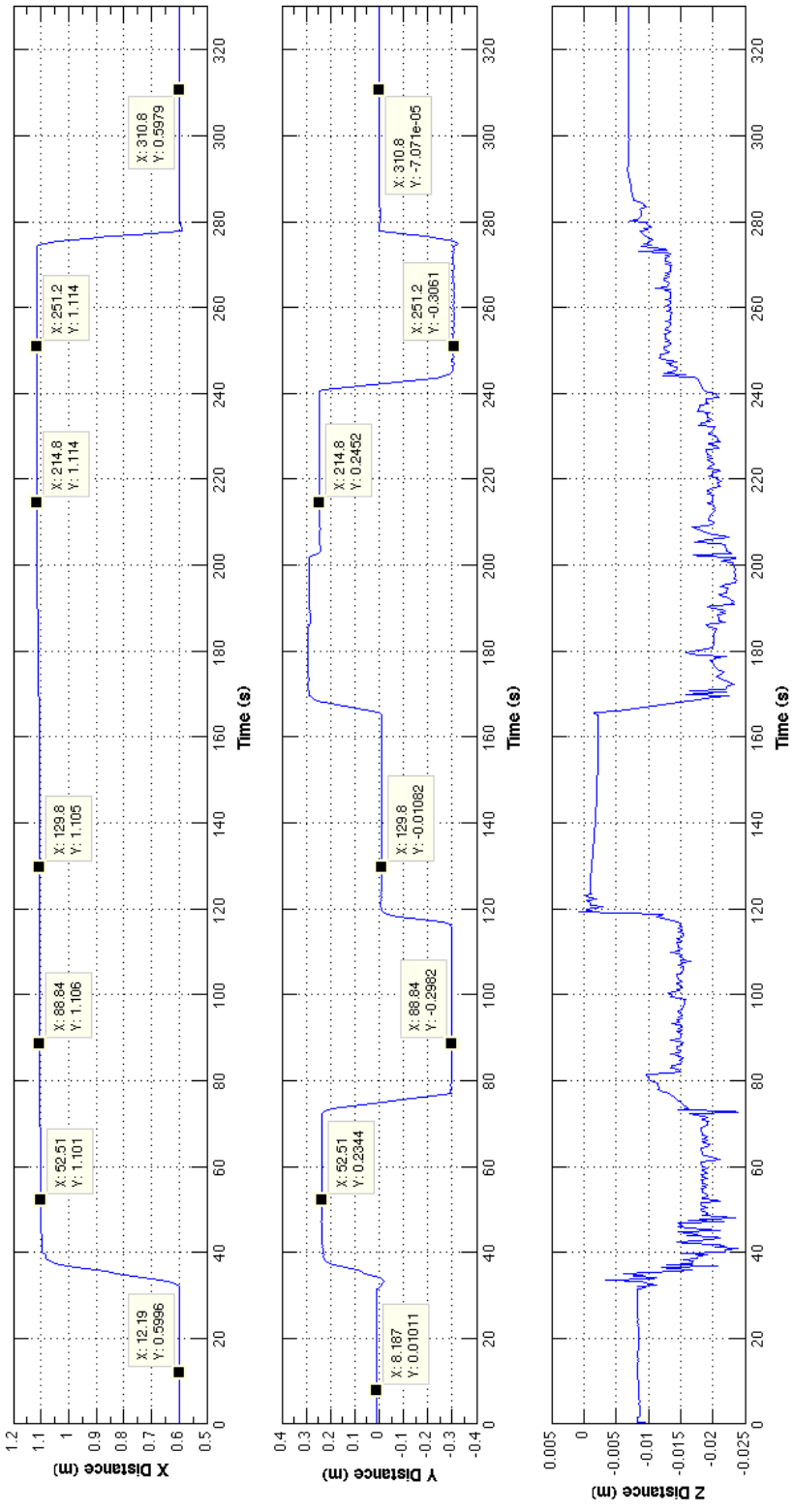


Figure 3-19: Relative Position Data From Exterior Orientation

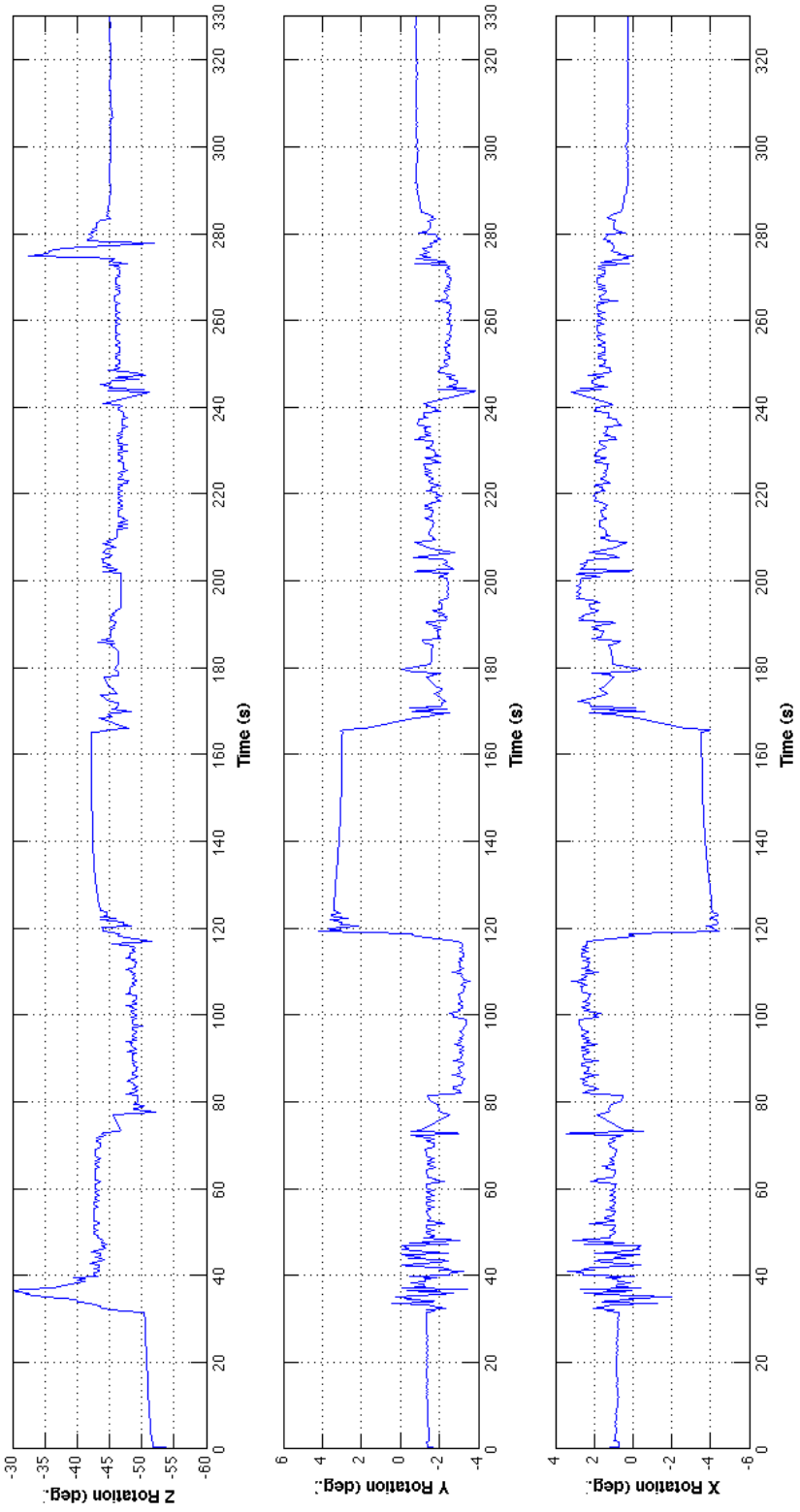


Figure 3-20: Relative Orientation Data From Exterior Orientation

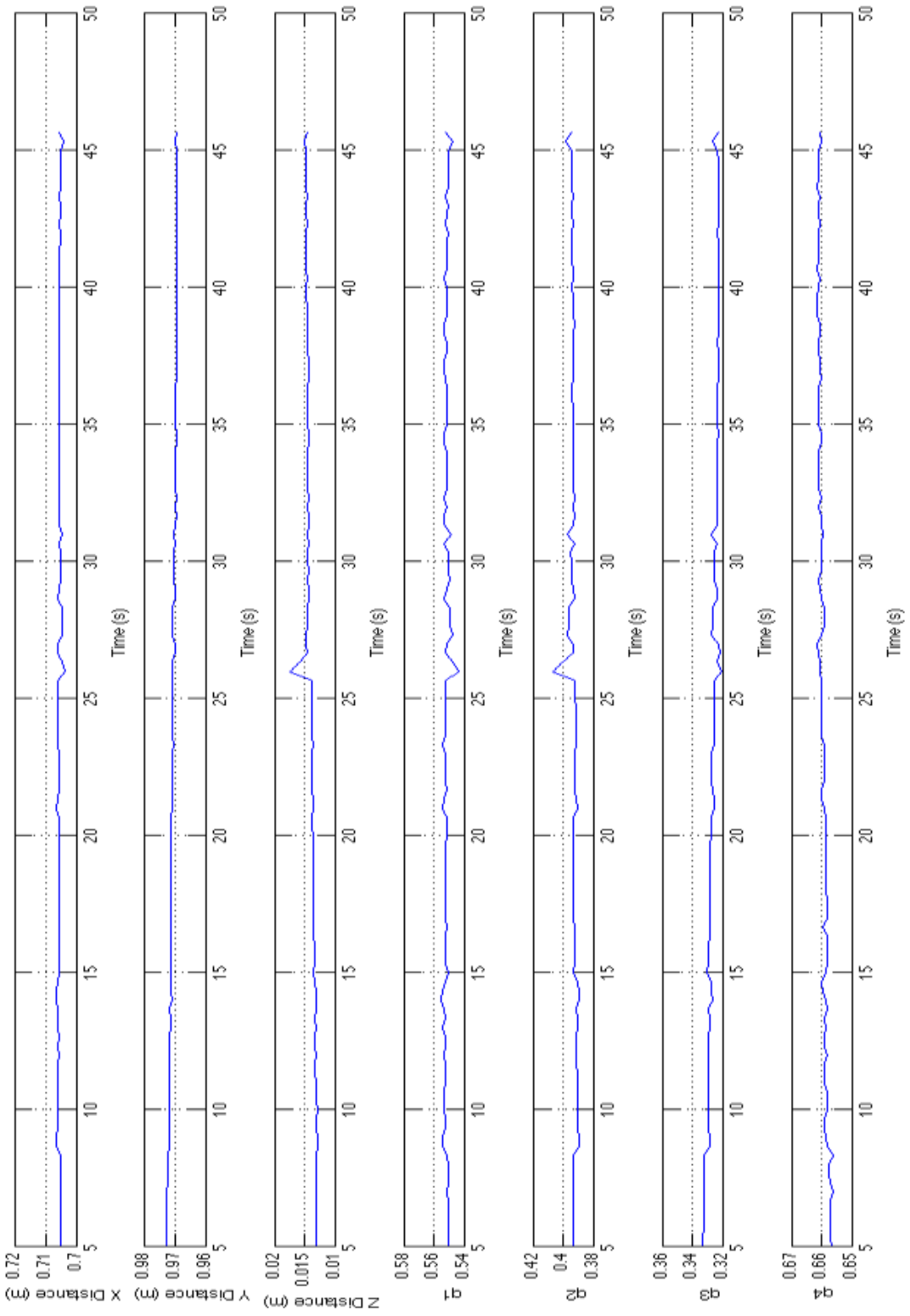


Figure 3-21: Converged Exterior Orientation with 2 Iterations per Timestep

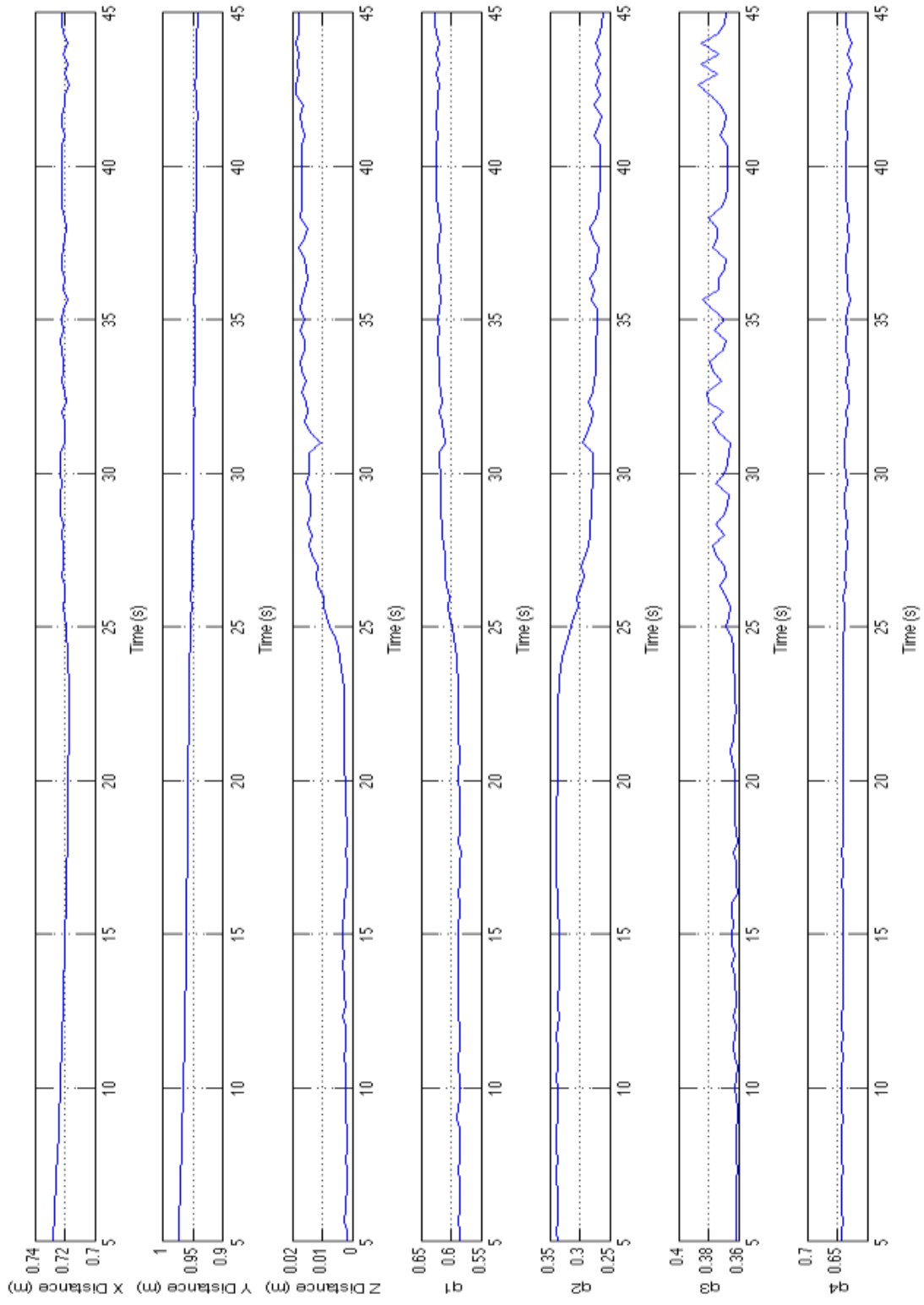


Figure 3-22: Converged Exterior Orientation with 50 Iterations per Timestep

3.7.3 Multiplicative Extended Kalman Filter Validation

The purpose of the Multiplicative Extended Kalman Filter (MEKF) is to filter the exterior orientation measurements with the relative vehicle dynamics and to estimate both the linear and angular velocities. Additionally, it provides an outlier rejection method and target re-acquisition system using the innovation's Mahalanobis distance.

In order to characterize the accuracy and precision of the output of the filter, the SPHERES performed a number of relative maneuvers that were measured by the vision system. These maneuvers were performed on the flat table using the SPHERES CO_2 thrusters (operating at 1 Hz), which were manually commanded using a joystick. Slight manual assistance needed to be provided in some cases to compensate for a slight tilt in the table and to overcome some small scratches in the table. It is important to note that in order to measure static positions, the air carriage's gas was turned off to take static measurements. This ensures the satellite doesn't drift over time while still floating on the table. The grid on the SPHERES flat table was again used as a ground truth for this comparison.

The maneuvers that were performed for validation of the MEKF are listed in Table 3.6.

Table 3.6: MEKF Validation Maneuvers

Test 1	Translation along the Camera's Z-Axis (depth)
Test 2	Translation along the Camera's Y-Axis (horizontal)
Test 3	Rotation of $\pm 45^o$ about the SPHERES Satellite's Z-Axis (vertical)
Test 4	High Angular Rate Rotation, Target Loss and Re-Acquisition
Test 5	Fault Detection and Outlier Rejection

Test 1: Translation along the Camera's Z-Axis (depth)

The first test showed a translation along the Camera'z Z (or Optical) Axis (the SPHERE 1 coordinate frame has been rotated by 45^o to make it easier to read by putting all of the translation in one axis). The relative X, Y and Z positions of SPHERE 2 relative to SPHERE 1 in this new frame are shown in Figure 3-23. The thrusters were used to translate SPHERE 2 exactly 30 cm away from SPHERE 1, the air carriage was turned off for approximately 50 seconds, then turned on again and the thrusters were used to return SPHERE 2 to its original starting position. Figure 3-23 shows that the static positions measurements of both the vision system and the SPHERES Global Estimator were within 5 millimeters of the ground truth as measured by the grid on the flat table. However, the SPHERES estimator appears to lag the vision estimator during the second translation, but not the first. This is due to the fact that the flat table is slightly tilted. The second translation was going "downhill" which was a faster process even though the exact same thruster commands were used. Since the SPHERES estimator does not model any significant disturbance forces (such as gravity in this case), while the vision estimator does, the vision estimator is likely closer to the truth.

The Y-axis of Figure 3-23 shows that the two estimators are within 1.5 centimeters of each other when the SPHERES are fixed in a static position, however the difference can be as much as 4 centimeters while the SPHERES are moving. In performing the maneuver, it was attempted to keep the Y-axis position as close to zero as possible, but it is entirely feasible that this manual regulation was only to within a few centimeters. As a result it is difficult to say which estimator is closer to the actual truth.

The Z-axis of Figure 3-23 shows a significant difference between the Z locations of the two SPHERES. Since this axis is in the vertical direction (i.e. perpendicular to the flat table) it is known that it should be zero. And any actual errors would likely come from differences in manufacturing of the air carriages, and curvature in the table. As a result, the fact that the global metrology system estimates that there is a 7 cm difference in the height of the two SPHERES is completely inaccurate. It is

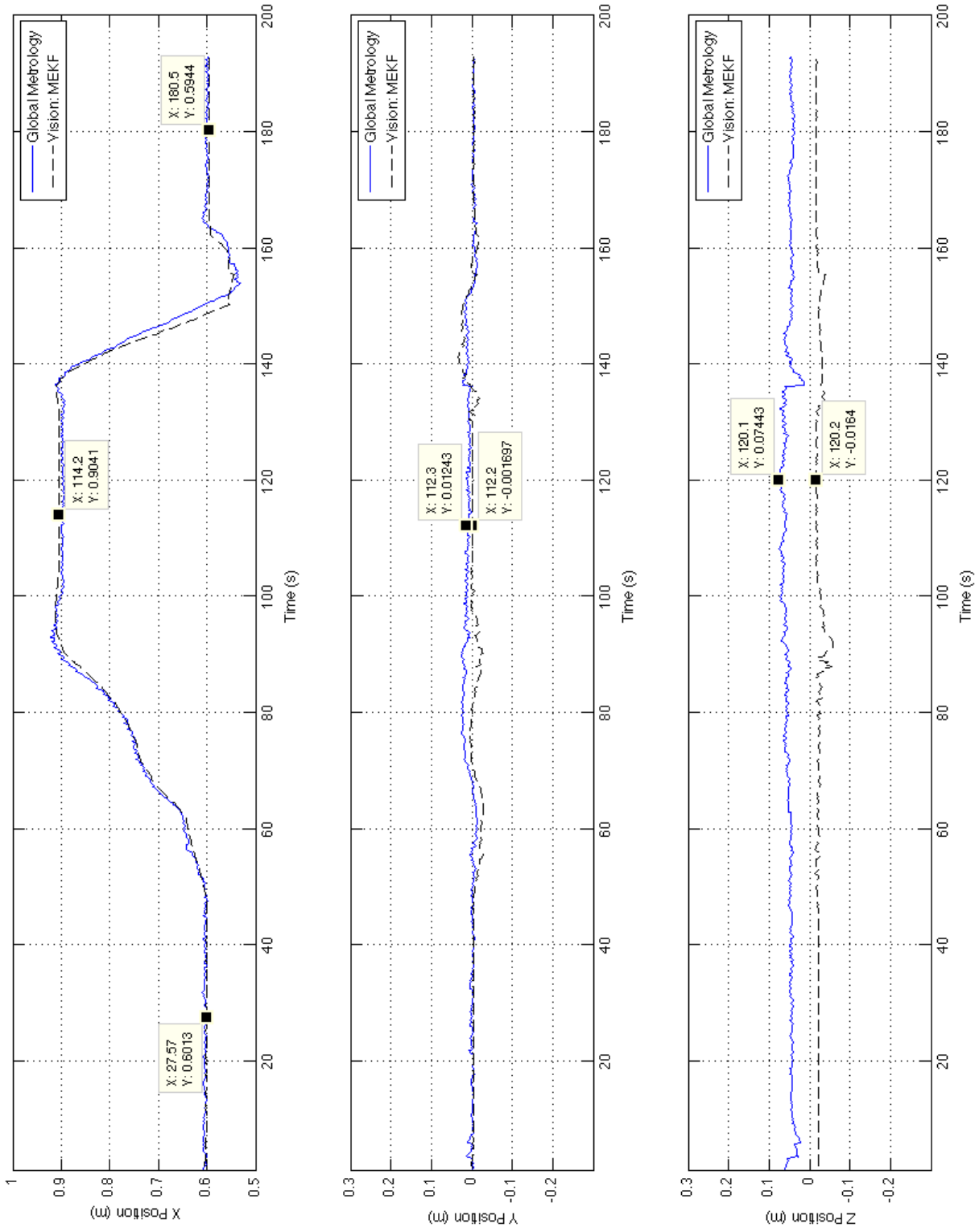


Figure 3-23: Relative Translation Along Camera's Optical Axis

believed that this is due to the fact that SPHERE 1 is incorrectly estimating is global orientation (possibly due to a sensor malfunction), and is biased by approximately a 7^o rotation about the Y-axis. This would cause the global metrology system to think it is pitched up, which would result in a difference in the relative Z-position of around 7 cm.

Figure 3-24 shows a close up version of Figure 3-23 between time 130 and 170. It also contains the raw sensor measurements from the exterior orientation algorithm, which are illustrated as red circles. Since the red circles correspond almost exactly to the black plus-symbols, this implies that the MEKF puts very high weight on the position measurements that are computed by the exterior orientation algorithm, which is what was expected based on Section 3.6.7. However, between 150 and 155 seconds, there were no measurements from the exterior orientation algorithm, due to the fact that the camera lost sight of the target. As a result, the MEKF just propagated the velocities for 5 seconds until it reacquired the target. This validates the accuracy of the dynamic propagation system, and the ability to reacquire a the target.

Using the slope of the line between 140 seconds and 150 seconds in Figure 3-24, it can be estimated that the global metrology system should estimate the velocity to be near 2.5 cm/s while the vision system should estimate the velocity to be near 3 cm/s. Figure 3-25 shows the linear velocity for a the same region of time. It is evident that the vision velocity estimate is much noisier than the global metrology due to the fact that there is a higher level of sensing noise that is being differentiated to determine velocity (see Appendix C.3 for a more detailed discussion of estimation of velocity from sequential position measurements). Although the measurements are noisy, they do agree with the above guess for velocity based on the linear slope of the position measurements between 140 and 150 seconds. It is important to note that the amplitude of the noise in velocity is greater when the SPHERE is moving. This is likely due to blurring in the image processing stage, which leads to incorrect estimates of the four targets. As a result it can be concluded that the velocity estimate has a precision of 2 mm/s and an accuracy of 2 mm/s in the static case, however in the

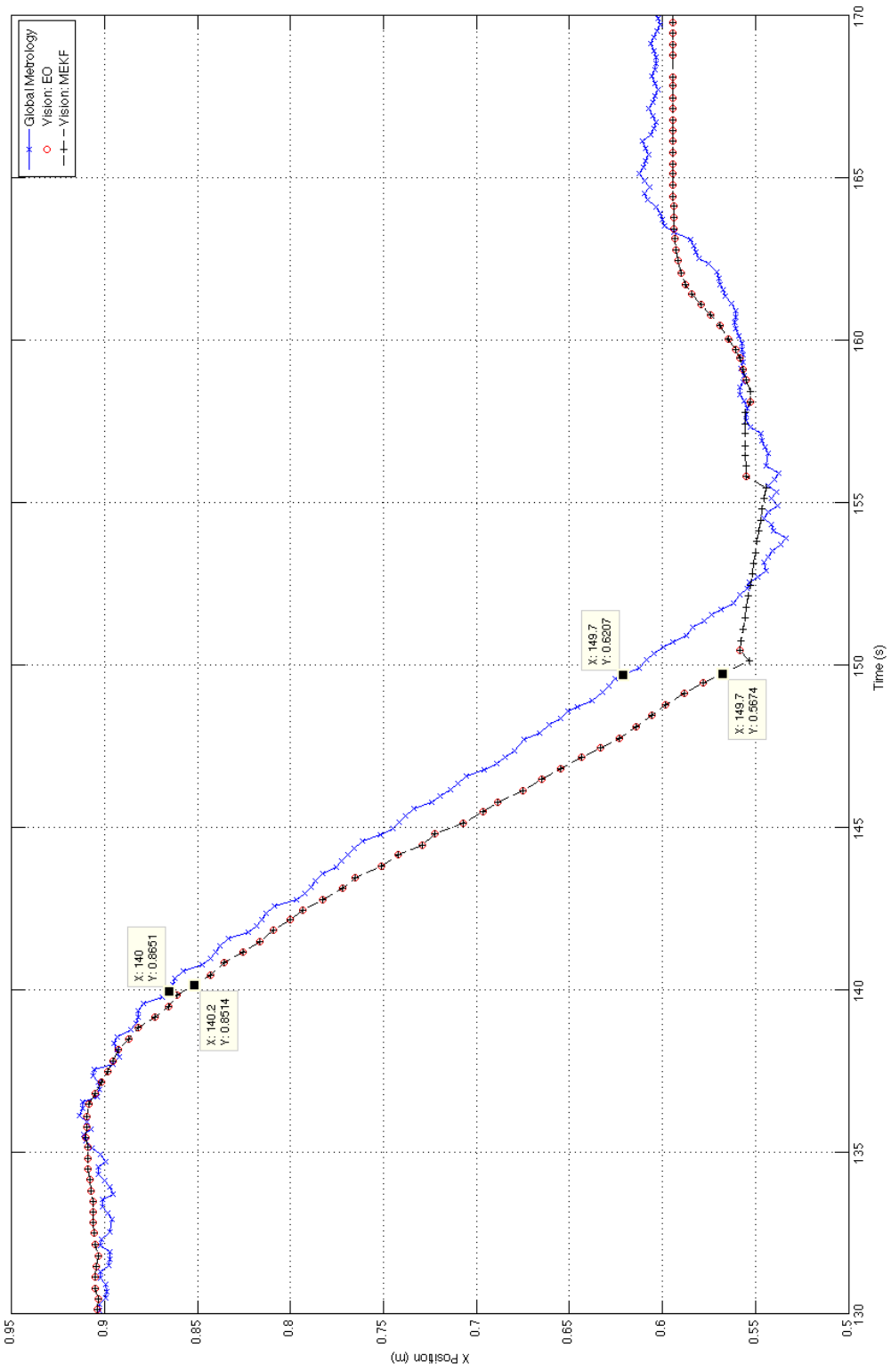


Figure 3-24: Zoom In On Relative Translation Along Camera's Optical Axis

dynamic case the precision is 2 cm/s and the accuracy is 0.5 cm/s.

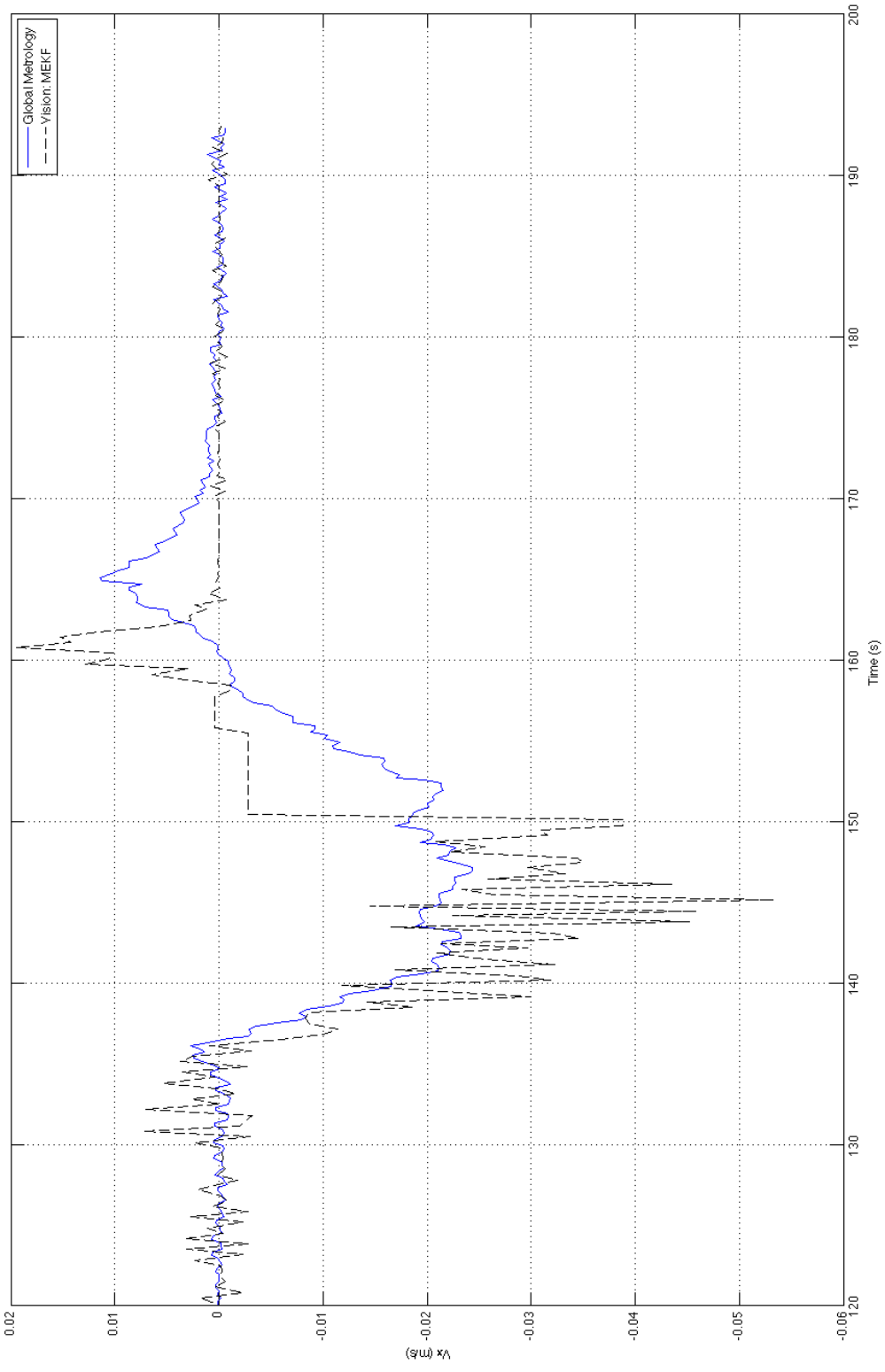


Figure 3-25: Zoom In On Relative Velocity Along Camera's Optical Axis

The covariances of the estimates are shown in Figure 3-26. When the target is in view, all of the estimates have a low covariance. However, 150 seconds and 160 seconds the target was lost (see Figure 3-23), and as a result the covariance grows quadratically for all of the states. Looking at the magnitude of the position covariance, when the target is in view shows that the standard deviation of the position is 0.05 mm. This is contrary to physical intuition given the previously discussed results. This indicates that the filter gives too much confidence to the position measurements. As a result, the covariance of the sensor noise could be tuned to a higher value in order to reduce some of the noise present in the velocity estimates.

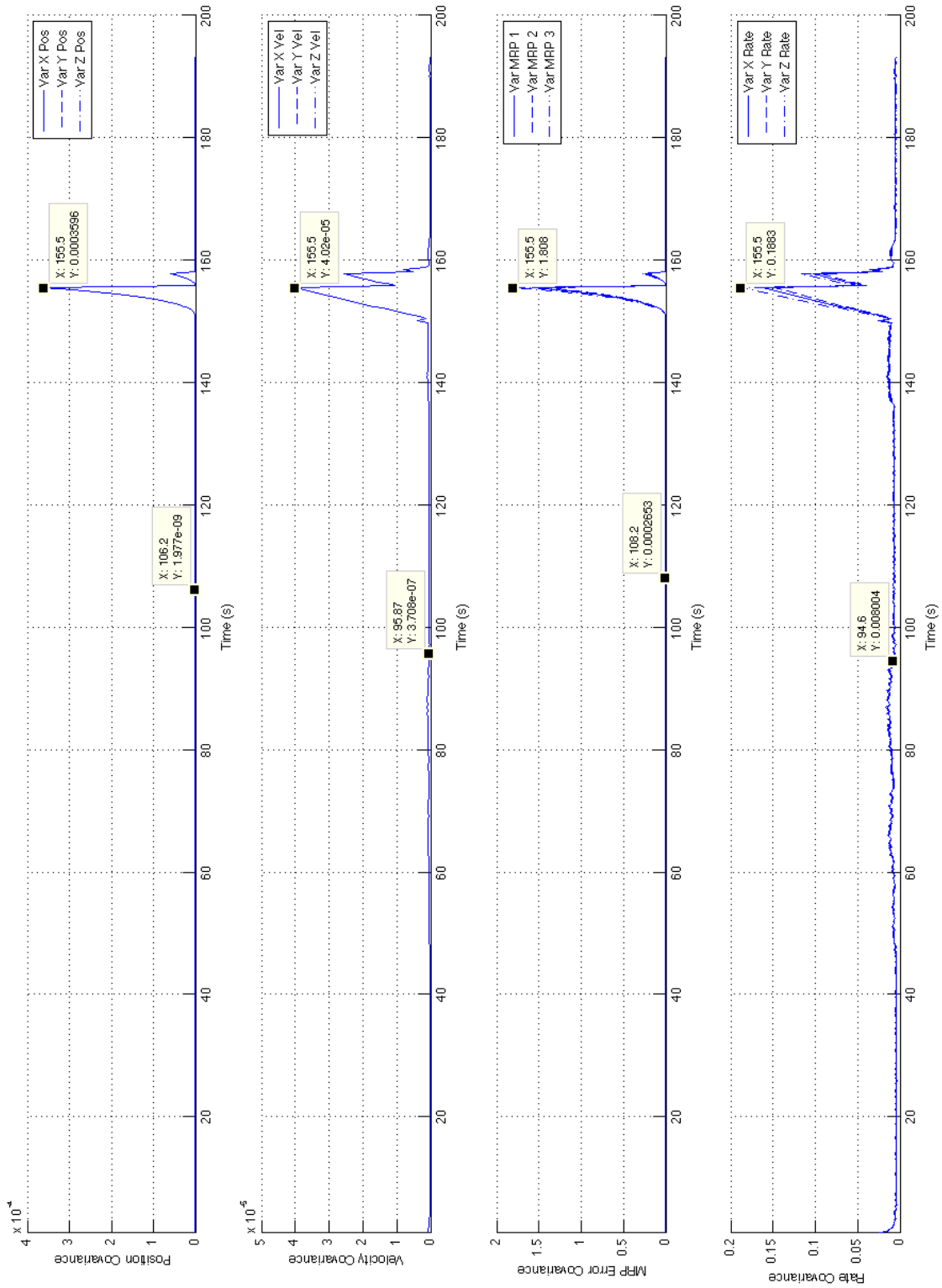


Figure 3-26: Covariance During Relative Translation Along Camera's Optical Axis

Test 2: Translation along the Camera's Y-Axis (horizontal)

In the next test, a translation maneuver was performed where the satellite traveled across the horizontal plane of the image (i.e. perpendicular to the optical axis). This is shown in Figure 3-27 (again the relative frame is rotated by 45^o). It shows that the difference between the global metrology and vision estimate of the X-axis remains within 0.5 cm of each other. The Y-axis shows the results of a 40 cm translation and return. As before, the global metrology estimate lags behind the vision estimate. Also, the global metrology consistently remains approximately 2 cm in the positive y direction from the vision estimate, which could indicate a bias in the coordinate frame transformations. As before, the Z-axis estimate of global metrology is definitely too large given that both SPHERES are sitting on the table. The vision estimate maintains approximately a -2 cm mean, which could also be due to a bias in the coordinate frame transformations. Due to the fact that the starting position of the SPHERE 2 was not near the initial conditions provided to the MEKF, it took a number of time steps for the estimate to converge. This is visible in the first few seconds of the state as well as in the covariance shown in Figure 3-28. This validates that the mean squared re-projection error from the exterior orientation stage is effectively being used as a metric for the certainty of the sensor measurement by the MEKF.

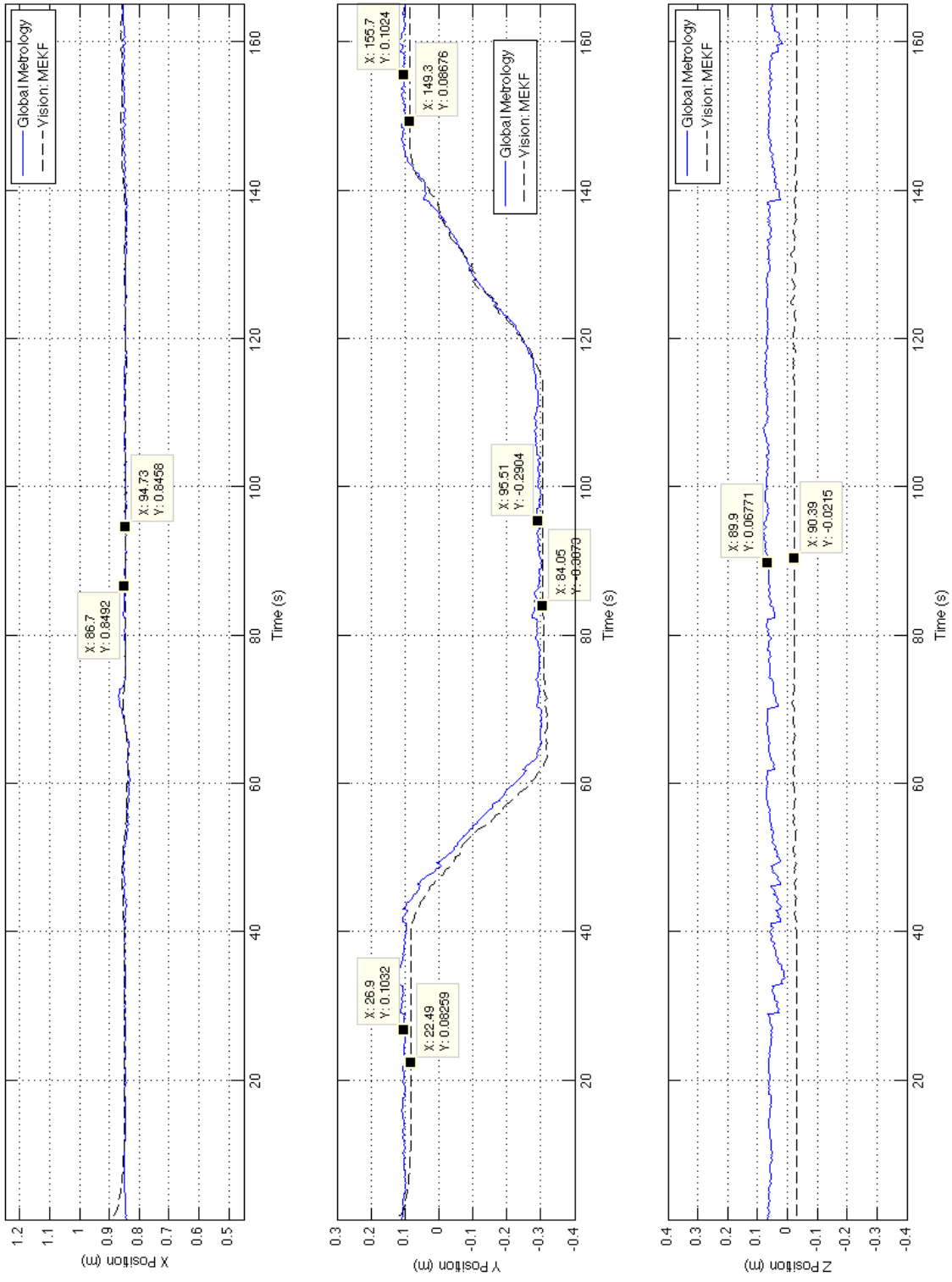


Figure 3-27: Relative Translation Perpendicular to Camera's Optical Axis

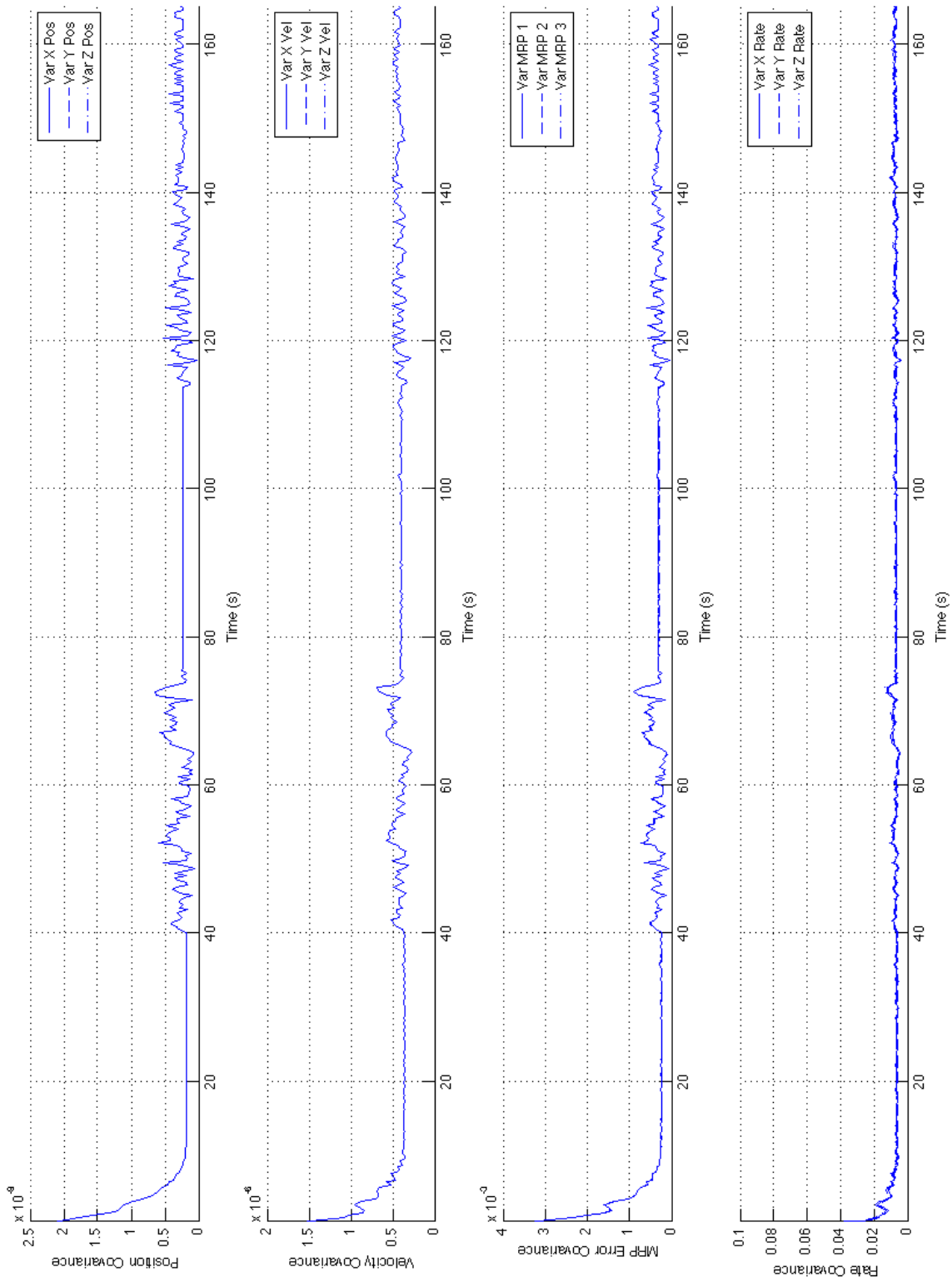


Figure 3-28: Covariance of Relative Translation Perpendicular to Camera's Optical Axis

Test 3: Rotation of $\pm 45^o$ about the SPHERES Satellite's Z-Axis (vertical)

A sequence of 45^o rotations were performed in the next test, which is shown in Figure 3-29. Throughout the test, the global and vision estimates of rotations about the X and Y axis remain within 4^o of each other. Manual placement of SPHERE 2 before turning off the air carriage was used to try to ensure each rotation was as close to 45^o as possible. However, each of the rotations that were found by the global and vision estimators are listed in Table 3.7. These results show that the difference between the global and vision estimates varied throughout the maneuver. In some cases the global metrology system gave a static measurement of the change that was closer to what was actually performed, while in other cases the vision did. This makes it difficult to determine which of the two systems better represent the actual orientation, in the dynamic case. Knowing what was the true orientation during dynamic rotation would help determine the accuracy. Unless other sensors are incorporated that are more accurate for measuring rotation, the upper bound on the dynamic orientation accuracy is $\pm$ 5 degrees.

Table 3.7: Sequence of Relative Rotations

Actual Rotation	**Change in Rotation**	**Change in Global Metrology Estimate**	**Change in Vision Estimate**
$-45^o \pm 2^o$	N/A	N/A	N/A
$-90^o \pm 2^o$	-45^o	-46.2^o	-40.4^o
$0^o \pm 2^o$	$+90^o$	$+90.8^o$	$+87.4^o$
$-45^o \pm 2^o$	-45^o	-39.8^o	-45.4^o
$-90^o \pm 2^o$	-45^o	-50.9^o	-45.1^o

In Figure 3-30 a close up view of the relative rotation is provided. This shows a distinct difference between the exterior orientation measurements (shown in a red circle) and the MEKF output. This means that the MEKF is actively weighting the predicted orientation in the resulting estimate, which shows that the filter is not just a pass through of the measurements from the exterior orientation stage. The differences in the exterior orientation and MEKF illustrate that the MEKF is acting as a low pass filter for the measurements using the dynamics of the two spacecraft system.

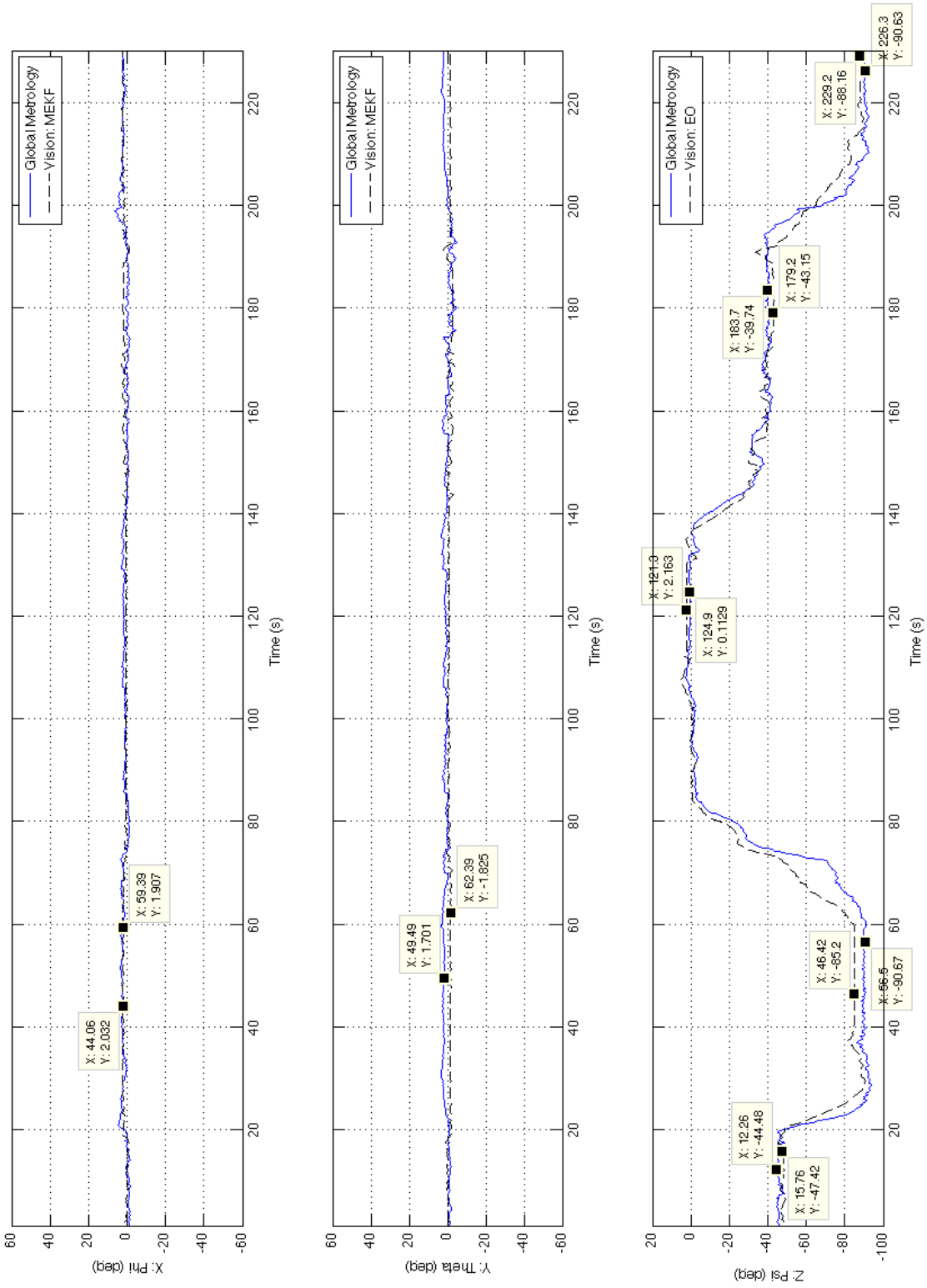


Figure 3-29: Euler Angles for Relative Rotation about Vertical Axis

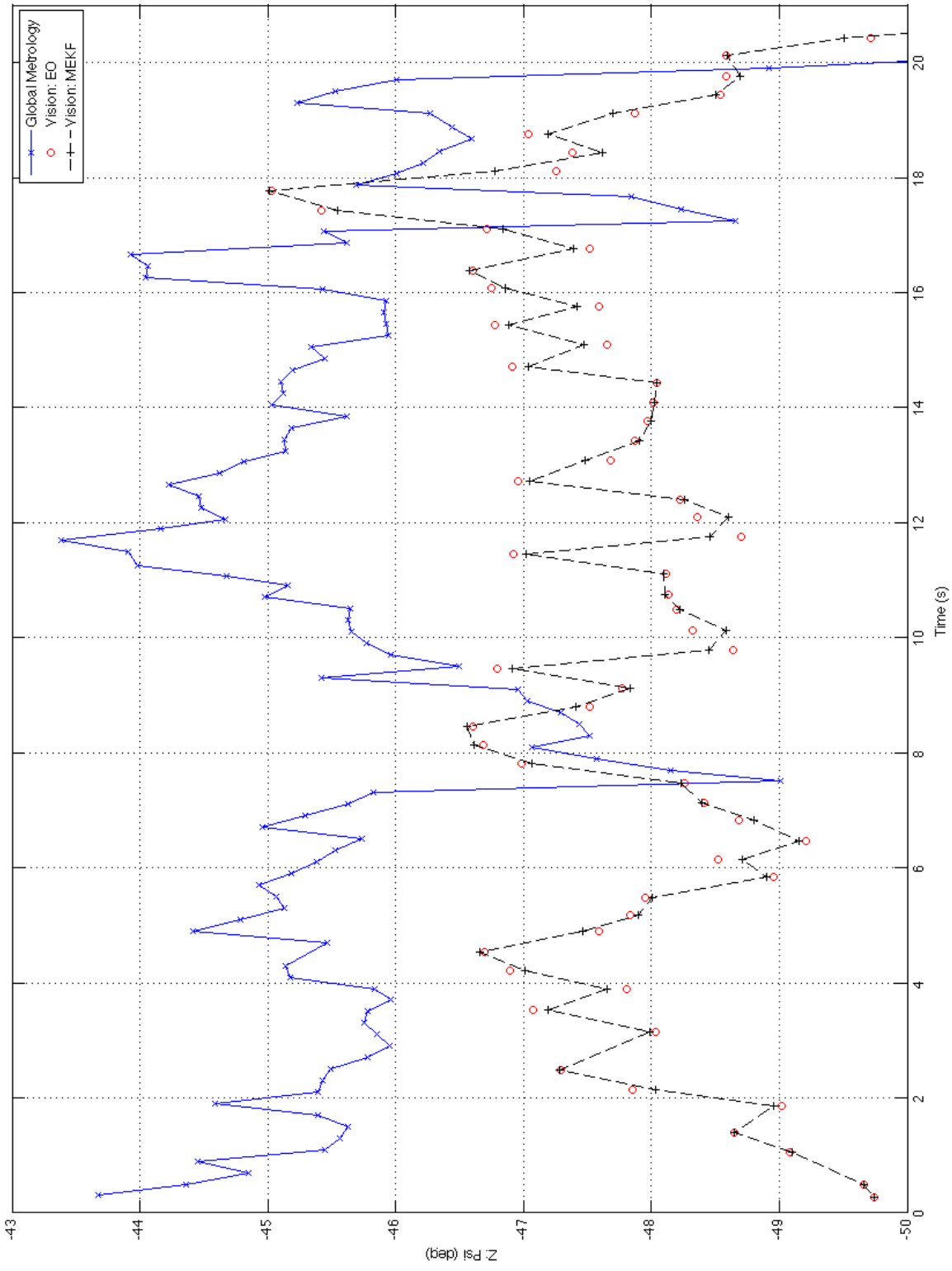


Figure 3-30: Zoomed in view of Relative Rotation

Test 4: High Angular Rate Rotation, Target Loss and Re-Acquisition

In the fourth test SPHERE 2 was rotated to the maximum extent that was detectable in one direction, and then a very high rotation rate was given in the opposite direction. Figure 3-31 shows that SPHERE 2 is first rotated to -90^o and then spun to 0^o before the target was no longer visible (red circles representing exterior orientation are no longer present), at which point the MEKF continues to propagate its estimate of orientation using the angular velocity. However, when the target is rotated back into the view of the camera the target is quickly re-acquired (red circles reappear in Figure 3-31).

The angular rotation rate is shown in Figure 3-32. The global metrology estimate shown is based heavily off of the gyroscopes that are onboard the SPHERES satellites and the integration of thruster commands. It is expected that this is a very accurate measurement of the angular velocity.

When the SPHERES are static (around the 10 second mark), they angular velocity estimate in both the global and vision estimators are close to zero. However when the SPHERES are moving, the angular velocity estimate of the vision estimator becomes very noisy (this is likely due to the fact that disturbances were being applied that were much larger than those that were modeled), however, the mean can be seen to track very closely to the global metrology estimate up until around 28 seconds. At this point it seems that the vision estimator corrected for a local minimum bias in the orientation estimate, given that at this angle (45^o) the camera has the best possible view of the target. At approximately 38 seconds, the camera looses sight of the target and the angular velocity remains constant until the target is reacquired around 44 seconds.

It is noteworthy that although the orientations did not match (Figure 3-31) during the period between 20 and 30 seconds (again due to a local minimum bias), the angular velocities tracked quite closely. This indicates that a very promising improvement to the system would be to use the angular velocity measurement of the vision system to estimate steady state gyro biases, while integrating the gyros to determine orientation.

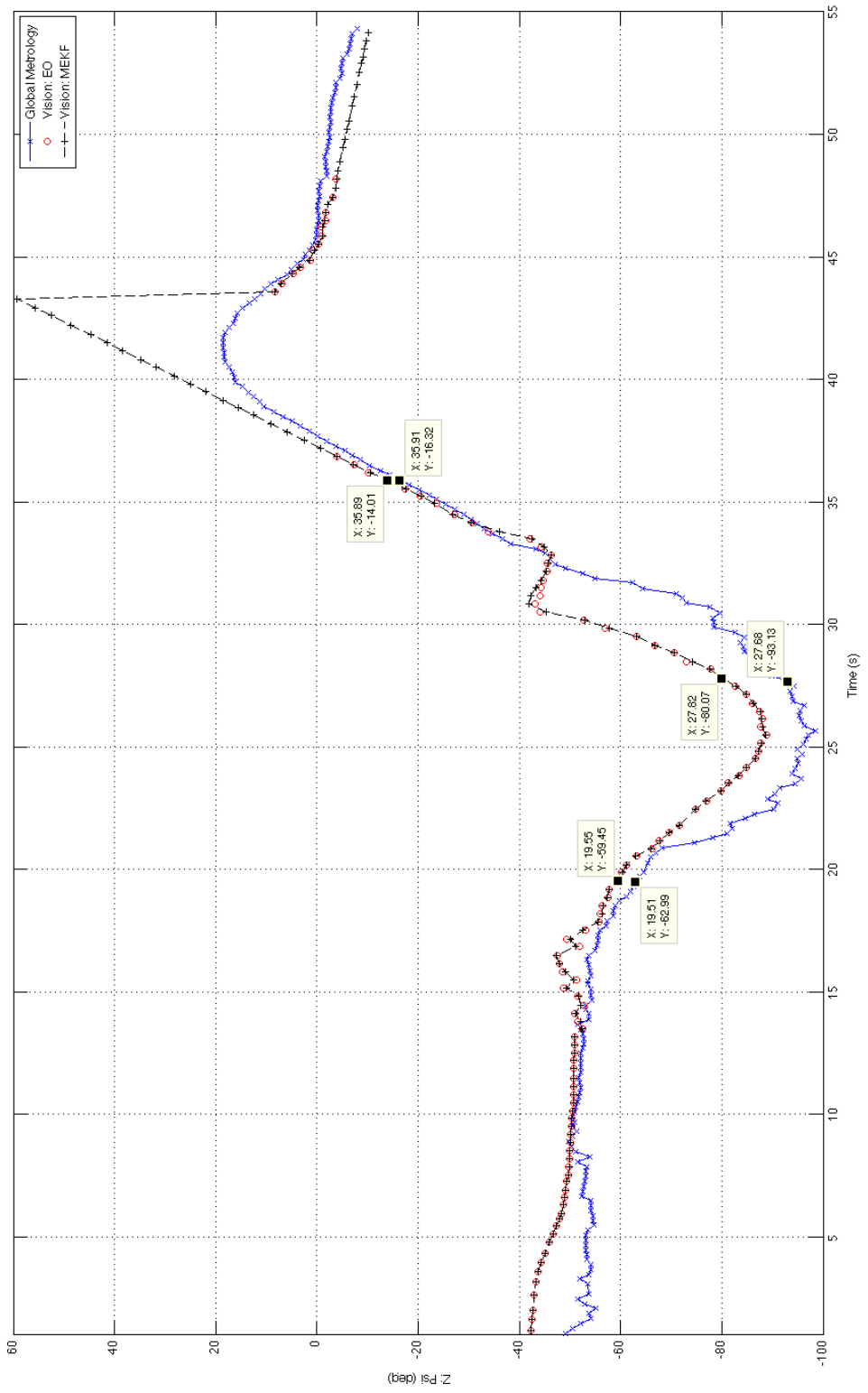


Figure 3-31: Relative Angular Rotation

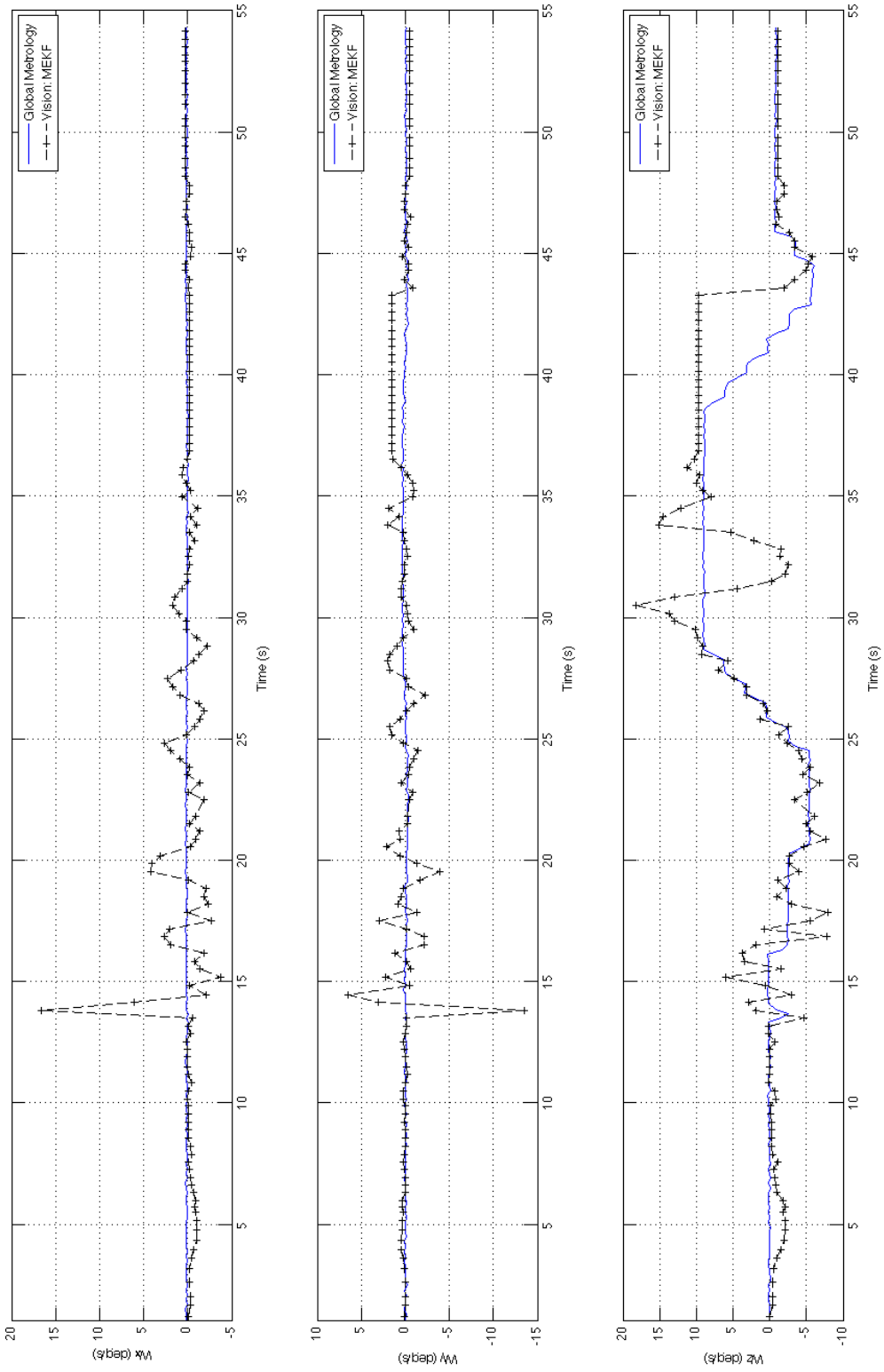


Figure 3-32: Relative Angular Velocity

Test 5: Fault Detection and Outlier Rejection

The purpose of the final test was to verify the operation of the fault detection and outlier rejection method (Section 3.6.6). In this test a target marker was visible by the camera and the MEKF was allowed to converge. Then, the marker was suddenly covered up, and a second marker was instantly visible at another location. The desired functionality of the outlier rejection method is to initially reject the new marker as an outlier, since the satellite should not be able to move between two locations that quickly. However, after a period of continuous rejection of the new target, it should eventually trust the new location and eventually accept it as the correct location.

The results of this test for Z-axis rotation are shown in Figure 3-33, along with the covariance of the Modified Rodrigues Parameters of the attitude error. At time 47.5, the old target was covered and the new target was made visible. For 4 time-steps, the output of the exterior orientation was rejected by the MEKF (the Vision estimate did not move from 45^o), and the covariance grew during this time. It eventually accepted a single measurement and estimated its full state. This significantly reduced the covariance. However, this new measurement implied a significant angular velocity, and since this is not the case, the subsequent measurements were rejected since they conflicted with the predicted value and the covariance grew again (i.e. the estimate overshot the correct value). Once the covariance was large enough that the measurements were not rejected the MEKF began accepting measurements again and the target was reacquired. This test was repeated with the exact same results at around 60 seconds. This verifies that the outlier rejection method described in Section 3.6.6 will reject outliers whenever there is high confidence in the current state estimate, but has a higher "relative" threshold for rejecting potential outliers when there is a lower confidence in the current state estimate. This allows for target reacquisition in the case that the satellite becomes "lost".

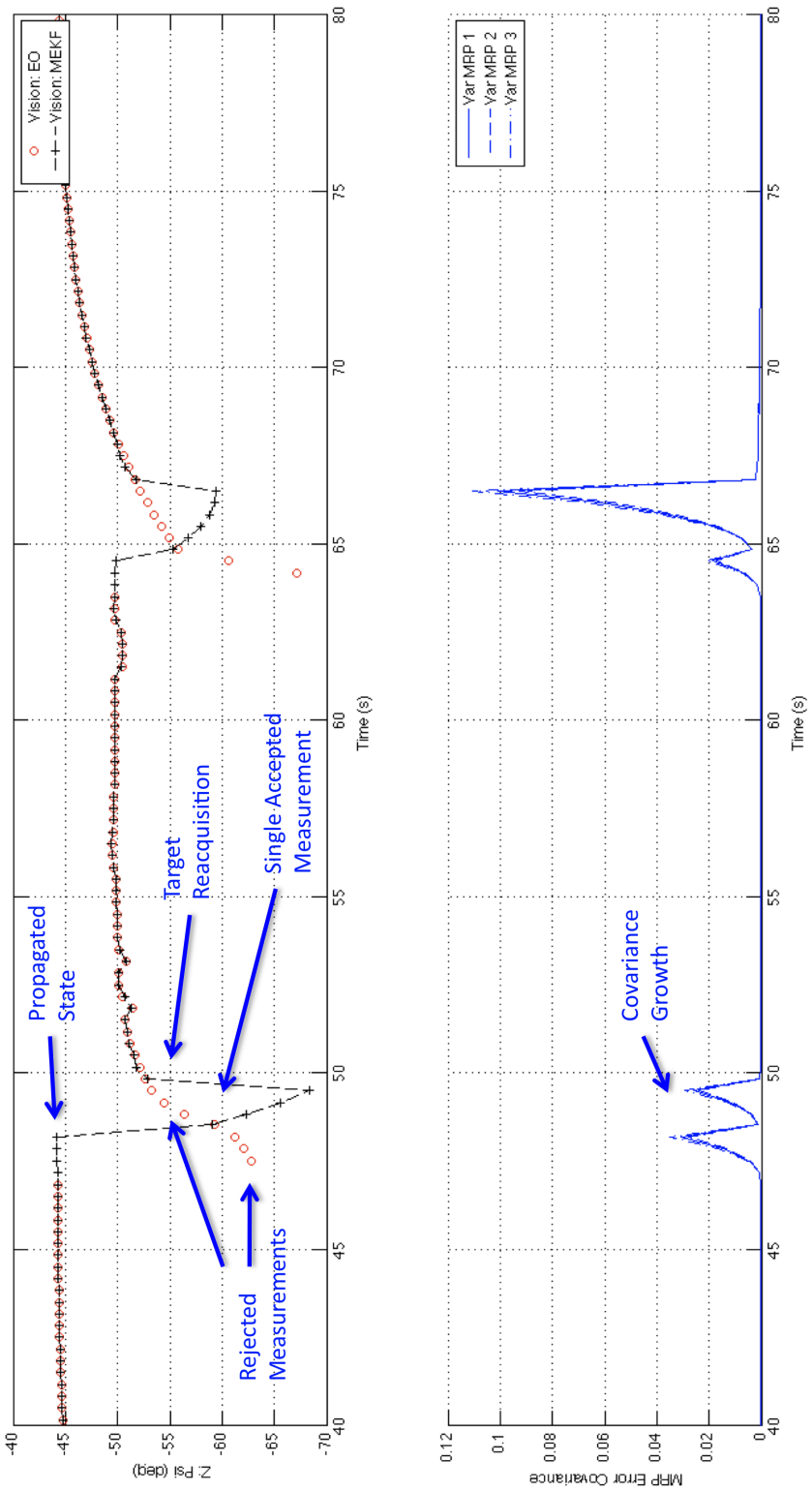


Figure 3-33: Outlier Rejection Test

3.8 Computational Requirements

In order to analyze the computational requirements that this approach has on the Goggles hardware, the execution times were measured for each of the stages described in this approach. The precision of the timing measurements is approximately 0.1 ms, due to the kernel implementation of *clock_gettime()*. However, given that there are a number of processes running simultaneously that can use up both cache and processor resources the accuracy of the measurements may vary by a few milliseconds. The data in Table 3.8 shows the execution times for each of the stages in the algorithm. Two times are shown for the Blob Extraction and Exterior Orientation stages, the first is when there is no initial guess given (the worst case time) and the second is when a good initial guess is given (the best case time).

Table 3.8: Execution Times for Algorithm Stages

Algorithm	**Runtime**
Image Undistortion	75 ms ± 1 ms
Segmentation	24 ms ± 0.3 ms
Connected Component Labeling (Entire Image)	98 ms ± 2 ms
Connected Component Labeling (Sub-Region)	71 ms ± 1 ms
CCC Target Filtering	0.8 ms ± 0.1 ms
Exterior Orientation (1000 Iterations)	37 ms ± 3 ms
Exterior Orientation (50 Iterations)	2 ms ± 0.2 ms
Multiplicative EKF	1.6 ms ± 0.2 ms
Total Execution Time (Best Case):	**172 ms**
Total Execution Time (Worst Case):	**243 ms**

The data in Table 3.8 shows that the Total Execution Time is dominated by tasks that need to access every element in an image. In the best case, these tasks account

for 168 ms of the total 172 ms (98% of the execution time). This is likely due to the fact that an entire image requires 300 kB of space, and the Pico-ITX computer only has 128 kB of on-chip cache memory. As a result there is likely a large number of cache misses that starve the processor of data.

Note that the image transfer from the USB camera to the Pico-ITX computer is implemented by the camera driver as a separate process. As a result, it is impossible to directly measure the time needed for the image transfer. Therefore extra buffer time is needed to allow for the image to be transfered to the computer. An update rate of 3 Hz was selected for this algorithm, which adds 90 milliseconds of extra time for the image transfered to take place.

3.9 Conclusions

This chapter presents an approach for relative navigation between two spacecraft using only computer vision information from a planar fiducial target. The details of the approach are described and experimental results using the SPHERES Goggles are discussed. From these results, it can be concluded that the system has upper bounds on precision and accuracies as described in Table 3.9.

Table 3.9: Upper Bounds on Accuracy and Precision of the Relative Navigation System

State	**Precision**	**Accuracy**
Static Position	± 0.5 cm	± 1.5 cm
Dynamic Position	± 1.0 cm	± 2 cm
Static Linear Velocity	± 0.1 cm/s	± 0.1 cm/s
Dynamic Linear Velocity	± 1.0 cm/s	± 0.25 cm/s
Static Orientation	± 0.5 degrees	± 3.0 degrees
Dynamic Orientation	± 2.0 cm/s	± 5.0 cm/s
Static Angular Rate	± 0.5 degree/s	± 0.1 degree/s
Dynamic Angular Rate	± 5.0 degree/s	± 1.0 degree/s

Benchmarks for the required computational time are presented, and it was concluded that this algorithm was best run on the SPHERES Goggles at an update rate

of 3 Hz. The computational time of the image processing stage was the dominant factor contributing to this update rate. Specifically, it was identified that cache misses due to a relatively small cache size likely caused the algorithms that traverse all of the pixels in the image to require significantly more computational time. This is an example of the memory subsystem (specifically the small cache) acting as a system bottleneck and starving the processor of data.

3.10 Future Work

Three main areas of future work for this system have been identified. The range of target detection could be increased, the computational time required for the image processing algorithms could be reduced and the exterior orientation measurements could be combined with additional information.

The operational range of the spacecraft relative navigation system is currently limited to an area of approximate one meter. This is due to the fact that the fiducial target was designed to fit onto the Velcro face of the SPHERES satellite and is only 7 cm by 7 cm, which combined with a 640 by 480 camera results in a small range of target acquisition. This range could likely be improved if either a larger target were designed or one with more detectable features. For example, when the target is lost, it is usually because the thinnest of the four concentric circles is broken. Additionally, bright light reflected off of the paper fiducial marker often leads to a loss of target.

As was previously discussed, the computational bottleneck is the memory system for the image processing system algorithms (e.g. image un-distortion, segmentation and connected component labeling). These algorithms were implemented by the OpenCV library and could be re-coded to improve performance on this particular system.

The last area of improvement is the inclusion of additional information in the MEKF. Currently it does not include any knowledge of control inputs. It is likely that in any spacecraft relative navigation scenario, the estimator would have access to the control inputs of at least one of the vehicles. Additionally, the estimates

of linear and angular velocity were shown to be noise, but very accurate. This is ideally suited to be combined with a set of inertial sensors such as accelerometers and gyroscopes which have a high precision, but whose accuracy can drift over time. In the MEKF, the bias of the gyroscopes and accelerometers could be estimated using the visual navigation system, while the position and orientation could be determined in the short term by integrating the inertial measurements with the estimated biases subtracted out.

Chapter 4

High Performance Embedded Computing for Vision Based Navigation within an Unknown Environment

4.1 Overview

The previous chapter has discussed the development of a vision-based relative spacecraft navigation system that requires the use of a fiducial marker of known geometry. However, in many applications, the environment is unknown and it is not possible to place a target with known geometry. A few examples missions where there is no prior knowledge of the environment are the inspection or servicing of an unknown or damaged satellite, an asteroid sample-return mission and terrain relative navigation for planetary landing.

A number of algorithms and approaches have been developed for navigation in an unknown environment (see Section 1.3.3 for a review). Specifically algorithms such as Visual Odometry (VO) or Simultaneous Localization and Mapping (SLAM) are often used to solve these types of problems. However, the implementation of these

algorithms in a small, low power, embedded aerospace system with a fast update rate is a challenge for many current and future applications that has not yet been solved. It is believed that one of the reasons for this is that there currently exists a mismatch between the algorithms and the computational hardware architecture on which the algorithms are run.

In this chapter, a preliminary study is shown that provides some justification for the belief that there is a mismatch between current processor architectures and vision based navigation algorithms for unknown environments. The reasons for why this mismatch exists are discussed, using a typical vision based navigation algorithm as an example (stereo vision depth map based on sum of squared distances). The characteristics of this example algorithm are analyzed in terms of parallelism and data access patterns. Suggestions are made for the type of hardware architecture that is best suited to run this algorithm. A comparison of existing computational hardware is presented where the primary figures of merit are giga-floating point operations per second per watt (GFLOPS/Watt) and memory/cache architecture. From this it is concluded that GPU's are likely to perform the best on the types of applications that are required for vision based navigation in an unknown environment.

These lessons are taken into account, and an approach to solve the problem of real-time visual navigation in an unknown environment with a small embedded system is proposed. This proposed approach implements dense, 3D occupancy grid, stereo-vision based Simultaneous Localization and Mapping on a Graphics Processing Unit.

4.2 Mismatch between Algorithms and Hardware

In Section 3.8, an analysis of the computational requirements of the fiducial tracking algorithm concluded that image processing steps that iterated over an entire image required much more time to complete than the other algorithms. This was likely due to the fact that the processor could not store the entire image in cache and as a result, the memory system was a bottleneck that starved the CPU for data.

This is an example of a situation where the algorithms and the computational

hardware are not working well together. In these types of data intensive algorithms it is reasonable to expect the overall performance to be very sensitive to changes in the memory system architecture. Another example of where this type of mismatch is believed to occur is on the Mars Exploration Rovers.

4.2.1 Mars Exploration Rovers

The Mars Exploration Rovers (MERs) are autonomous systems that operate in an unknown environment and are very sensitive to power, mass and volume constraints. These rovers have a top mechanical speed of 124 meters per hour, however when the autonomous software is activated, the rovers slow down due to the extra time that is needed for processing [32]. The relative speeds for the rovers when different software systems are enabled are shown in Table 4.1 [64, 8]. The MER rovers use a 20 MHz RAD6000 CPU that runs VxWorks and has 128 MB of RAM and 8 KB of on-chip cache. Since it runs all software systems on the rover, its memory space and computational resources are shared by 97 other tasks.

Table 4.1: Driving Speeds for Mars Exploration Rovers

Mode	**Speed**
Manual Driving	124 m/hr
AutoNav (safe terrain)	36 m/hr
AutoNav (obstacles)	10 m/hr
Visual Odometry	10 m/hr
Visual Odometry + AutoNav	5 m/hr

Figure 4-1 shows performance data from [32]. It shows timing results for each instruction of the underlying "inner loop" sub-algorithms for the stereo vision correlation step that is used on the MER rovers. Both the C Code (blue) and Vector Code (red) were run on the same 700 MHz Pentium III computer (which was running windows 2000 with 32 KB L1, 256 KB L2 and 512 MB main memory). The main

change in the vector code was the inclusion of a prefetch instruction, which specifies which data is likely to be used in the future. The data shows much larger spikes in the unoptimized C code, which implies that the computational performance is limited by the memory subsystem. This indicates that modern desktop processors will not be as well suited to implement the types of algorithms that are used on the MER rovers.

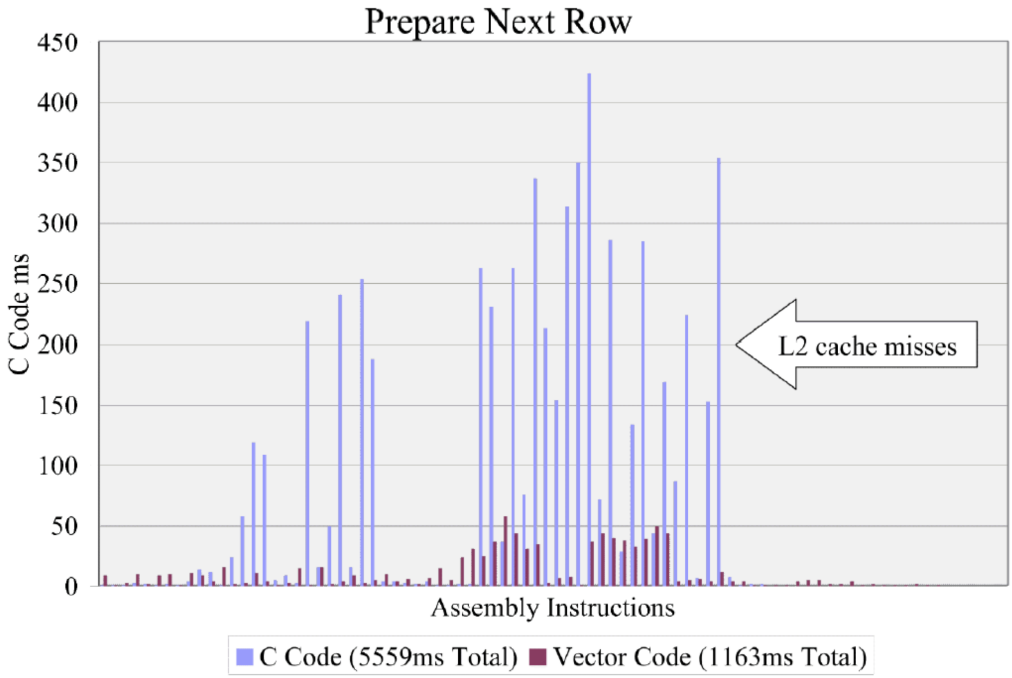


Figure 4-1: MER Inner Loop Execution Time per Assembly Instruction over 200 Iterations [32]

This illustrates that there is a mismatch between the algorithms and hardware that has an effect on the overall capabilities of the system (i.e. the drive speed) and ultimately the scientific capabilities of the MER rovers.

4.3 Computational Characteristics of a Stereo Vision Algorithm

In order to properly match software algorithms to hardware architectures, it is important to fully understand the computational requirements of the algorithm. The first step to characterizing the computational requirements of vision-based navigation algorithms is to analyze a commonly used, prototypical algorithm. Stereo vision was selected as a common computer vision method, and is discussed in this section. It is used for determining a depth map of a scene using two cameras with a known baseline separation [32]. The most common method of performing this computation is the "sum of squared differences" (SSD) algorithm. The main iterative loop of the algorithm for n by m left and right image ($I_L(x, y)$ and $I_R(x, y)$) is shown in Algorithm 2, which outputs a depth map $D(x, y)$.

Algorithm 2 Pseudocode for Stereo Correlation Algorithm: Sum of Squared Differences

for $x = 1$ to n **do**
  for $y = 1$ to m **do**
    for $d = 1$ to d_{max} **do**
      for $w_x = -w$ to $x + w$ **do**
        for $w_y = -w$ to w **do**
          $B(d) \leftarrow B(d) + (I_L(x + w_x + d, y + w_y) - I_R(x + w_x, y + w_y))^2$
        end for
      end for
    end for
    $D(x, y) \leftarrow \arg\min_d B(d)$
  end for
end for

This algorithm has a number of computational properties that can be exploited with the appropriate hardware. They are listed below:

1. The first two FOR loops are data parallel

2. The only branching that occurs is in the minimization step.

3. The instructions complexity is fairly low. Only floating point add/subtract, square and compare are required (presuming the for loops are unrolled).

4. The memory array $I(x, y)$ is always read from, while the depth map $D(x, y)$ is always written to.

5. The inner-most two loops have spatial locality in two-dimensions rather than one.

The specific features of the computational hardware that are needed to take advantage of each of these properties can now be discussed.

Item 1 states that the algorithm has significant data parallelism. In order to exploit this fully, the hardware must have a large number of floating point units. The computational time of the parallel stereo vision algorithm has the complexity $O(\frac{nm}{p})$, where p is the number of parallel floating point units.

Item 2 listed above indicates that there is very little branching in the algorithm. As a result it is probably not necessary to have a branch predict unit, which often requires a significant amount of power in modern processors.

Item 3 suggests that only a minimalist instruction set architecture is necessary and it is not necessary to consume extra power to decode more complex instructions.

Caches typically incorporate circuitry that is designed to synchronize multiple copies of the same data. Item 4 implies that this is not necessary, since the data is primarily only read from or written to. This could be further this could be extended to the fact that the depth map $D(x, y)$ does not need to be stored in a cache at all during the stereo algorithm.

Item 5 leads to one of the most promising optimizations for image processing operations. Typically, caches will load linear blocks of memory that are near the element currently being accessed in the hopes that these nearby elements will be accessed in the future. In this image processing application, this implies that rows (or subsets of rows) of the image are loaded into the cache. However, the stereo vision algorithm often needs values that are one column above or below. Caching the

memory in two dimensional blocks would likely allow for a lot fewer capacity misses in the cache system.

4.4 Comparison of Processors

Using the characteristics discussed in the previous section, a comparison of different processing architectures can now be performed. In order to predict which processor would have the best real-world tradeoff between performance and power, mass and volume, two figures of merit are used. The first is the theoretical peak Giga-Floating Point Operations per Second per Watt (GFLOPS/Watt). This is used because it represents the trade between computational performance and power consumption (which ultimately drives mass and volume of the embedded system). However this theoretical peak GFLOPS is only achievable if the memory sub-system can supply it with enough data so that the computational units are not sitting idle. Therefore, the second metric is a the memory architecture, which is more subjective, but gives an indication of how close a particular processor will be able to come to its peak theoretical GFLOPS/Watt. Table 4.2 shows a comparison of different processors using these figures of merit.

The CPU's listed in this table, such as the Xeon and Core 2 Duo, do not compare well in terms of GFLOPS/Watt. From this it can be concluded that for any data-parallel application, standard CPU's are not the best architecture choice in terms of GFLOPS/Watt. The reason for this is that many typical CPU's spend a significant amount of power on sub-systems that are not necessary for vision based navigation, according to the characteristics listed in Section 4.3. For example, a Pentium 4 Williamette, which runs at 1.6 GHz and requires a total power of 60.8 Watts, will only spend 3.6 W on floating point operations. However, this chip will spend 11.0W on bus control, 10.0 W on instruction decoding, 6.1 W on dynamic translation lookaside buffer, and 22.3 W for idle operations [99].

Table 4.2: Processor Comparison

Processor	**Theoretical Peak GFLOPS**	**Watts**	**GFLOPS/ Watt**	Memory Architecture
Quad "Bloomfield" Xeon 3.2 GHz	25.6 GFLOPS	130 W	0.197	Shared Linear Cache
Core 2 Duo "Penryn" 2.53 GHz	20.2 GFLOPS	25 W	0.808	Shared Linear Cache
Cell Processor	152 GFLOPS	80 W	1.900	Shared Linear and Shared Programmable Cache
NVIDIA Tesla C870	518 GFLOPS	170 W	3.047	Shared Programmable Cache and Two-Dimensional (Texture) Cache
NVIDIA GeForce 9800GT	504 GFLOPS	105 W	4.800	Shared Programmable Cache and Two-Dimensional (Texture) Cache
NVIDIA GeForce 8800M	240 GFLOPS	35 W	6.857	Shared Programmable Cache and Two-Dimensional (Texture) Cache
Tilera TilePro64	221 GFLOPS	23W	9.609	Message Passing and Shared Linear Cache

This table does not include Field Programmable Gate Arrays (FPGAs) due to the fact that their GFLOPS/Watt is highly dependent on the actual programmed architecture. FPGAs have been used for a number of hardware acceleration systems for space flight applications[69], but need to be programmed using a hardware description language such as VHDL or Verilog.

The processors with the best GFLOPS/Watt are the Tilera processor and the NVIDIA Graphics Processing Units (GPU's). The Tilera processor would be ideally suited for any embedded data parallel application, however some authors have found difficulties mapping the message passing architecture to real world navigation applications [93]. Current publications have shown that the GPU's have had significant success in accelerating real world computer vision applications [30, 66]. This is likely due to the fact that the texture caches are shown to utilize a two dimensional principle of locality, and caches data in 2D blocks [36].

Assuming all of the floating point units on a GPU can be fully utilized, the GeForce 8800M listed in this table would increase the processing capability by a factor of 35 compared to the Quad Xeon processor for the same electrical power consumption.

Given this analysis for the stereo vision algorithm, and that fact that many of these arguments can be extended to a number of vision based navigation algorithms, this indicates that GPUs are likely to be successful in accelerating other vision-based navigation systems in a power efficient manner. However, there are always drawbacks to any design decision. The tradeoffs associated with using any sort of parallel hardware architecture such as a GPU, is that the programming becomes much more difficult. Parallel programming techniques open the door to race conditions and deadlocks, which require the proper use of synchronization and mutual exclusion. Any guarantees of correctness (which would be necessary for many high profile space missions) are difficult to make when the order of execution is not completely deterministic. In the past, GPU's have been very difficult to program since the only way to do this was by mapping the program to an OpenGL implementation. Recently, C programming API's such as NVIDIA's Cuda or OpenCL have allowed GPU's to be programmed in C [13], which has resulted in significant research attention in recent years.

4.5 Proposed Implementation of GPU-Accelerated Vision Based Simultaneous Localization and 3D Occupancy Grid Mapping

The preceding analysis of the computational performance of a SSD stereo vision algorithm for various hardware architectures has shown that GPU's would likely be best suited for an real-time embedded system. Although this analysis only considered a stereo vision algorithm, it is hypothesized that the same logic can be applied to all of the necessary stages for a full visual navigation system that would be capable of operating in an unknown environment. In this section a potential approach is proposed that uses Simultaneous Localization and Mapping (SLAM, see Section 1.3.3 for details).

The proposed method would develop a real-time Visual SLAM algorithm that represents the environment using a three dimensional occupancy grid map. Current visual SLAM algorithms are only able to create sparse maps of the environment, which are insufficient for path planning [16]. A 3D occupancy grid map would enable path planning in a complex three dimensional environment, that would be necessary for the use of vision based navigation on missions such as asteroid sample returns.

The block flow diagram of this approach is shown in Figure 4-2. In this approach, the images will be captured, preprocessed and transferred to the GPU at the beginning of each time step. Once this is done, all of the estimation and path planning would be performed onboard the GPU. A stereo camera configuration would be used (rather than a monocular system) to obtain a depth map in a computationally efficient manner.

It should be noted that this approach is very similar to the approach to visual navigation with a fiducial marker discussed in Chapter 3. This approach begins with an image preprocessing stage (e.g. undistortion), followed by an image processing stage, which is similar to CCC Detection. Next a nonlinear minimization algorithm is used, such as Iterative Closest Point (ICP), which is very similar to exterior orien-

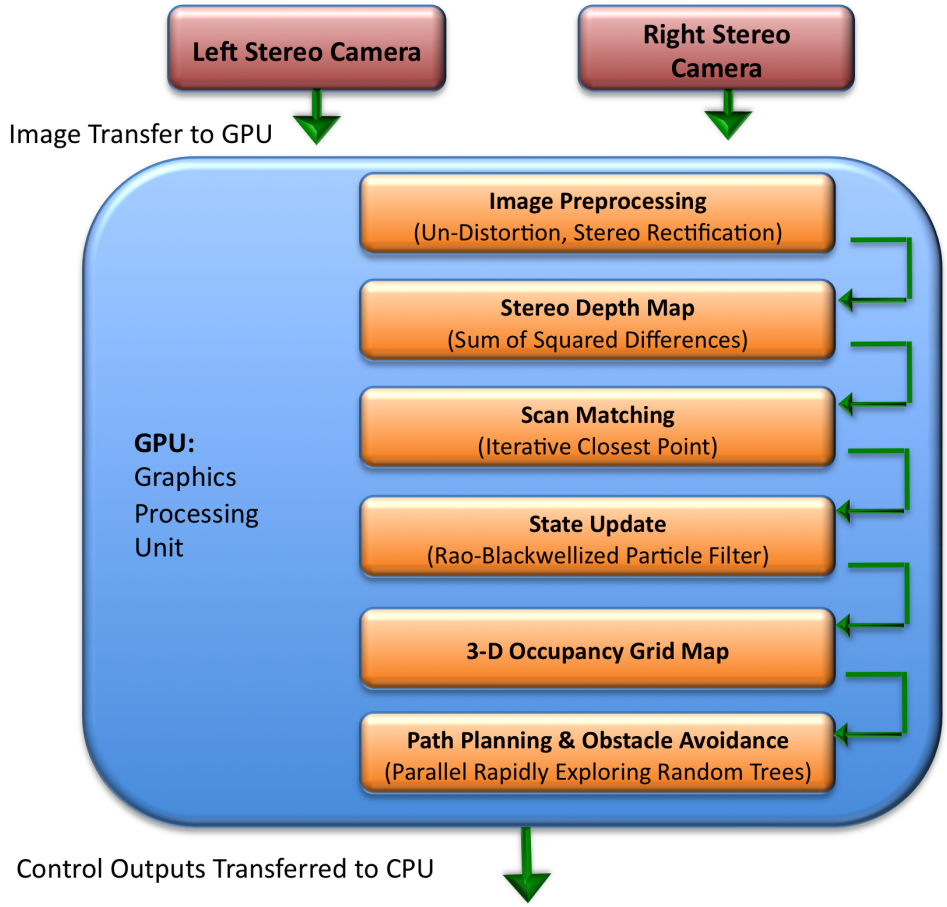


Figure 4-2: Proposed Stereo Vision Based 3D Occupancy Grid Approach on a GPU

tation. This is followed by a recursive state estimator such as a particle filter, which performs a similar function to the MEKF previously discussed. The construction of an occupancy grid map and the path planning within this map does not have an analog to the previously discussed fiducial marker tracking.

This approach is also similar to existing occupancy grid SLAM algorithm that use two dimensional LIDAR sensors[87]. The two main changes are than stereo vision is used to calculate the range information and that the method will need to be extended to three dimensions. Both stereo vision and occupancy grid SLAM have been been investigated by a number of researchers, however the primary innovation of this approach would be to combine these methods in a manner that achieves the maximum performance when implemented on a GPU.

The algorithms for each of the stages shown in Figure 4-2 have been individ-

ually selected based on their ability to be parallelized and accelerated by a GPU. In a preliminary study, the stereo depth map algorithm has been implemented and benchmarked against a published CPU implementation [92]. The GPU accelerated method was able to compute a density map for a 640 by 480 image in 25 milliseconds on a NVIDIA GeForce 9800GT [90], whereas the CPU implementation required 100 milliseconds for a 512 by 512 density map. This implementation took advantage of SIMD vector acceleration using SSE2 instructions on a Pentium 4, 3.2 GHz processor.

All of the other algorithms selected and shown in Figure 4-2 have parallel implementations that have been discussed in literature and therefore should map well to a GPU. Scan matching algorithms are used to align a set of points with a known map, and are very similar to the absolute orientation problem. A parallel version of this algorithm has been presented in [55]. The Rao-Blackwellized Particle Filter is a method for estimating the probability distribution of a state based on a Hidden Markov Model. Since it is based on sampling and propagating a sufficiently large number of Monte-Carlo samples at each time step, it should be much easier to parallelize than an Extended Kalman Filter. The EKFs most computationally demanding task is matrix inversion (which is not as easily parallelized). A number of authors have published results on parallel particle filter implementations, including on GPUs[12, 41]. Lastly a path planning and obstacle avoidance system that can be parallelized is needed. Methods for parallel rapidly exploring randomized trees (RRT) have been previously published by Carpin[15].

The existence of parallel algorithms for a number of these methods shows that this approach has potential and should be considered for future research and experimental implementation.

4.6 Summary and Conclusion

To summarize, this chapter argues that there is a mismatch between current vision based navigation algorithms for unknown environments and the embedded computational hardware that is used to implement these algorithms. A few examples, such

as the Mars Exploration Rovers, are given to support this argument. The computational characteristics of a prototypical vision based navigation algorithm (SSD Stereo Vision) are discussed. Using these characteristics, different types of processing architectures are compared in terms of GFLOPS/Watt and memory system architecture. This comparison concludes that Graphics Processing Units are best-suited implement vision based navigation algorithms in an embedded system. This is due to the parallel processing and two-dimensional cache architecture of GPUs. The tradeoff of this approach that was identified is the software engineering complexity involved with implementing data parallel algorithms.

A direction of future work was identified that implements a GPU accelerated, stereo vision based simultaneous localization and mapping system, which build a dense 3D occupancy grid for use with a parallel RRT-based path planning and collision avoidance system.

Chapter 5

Conclusions

5.1 Summary of Thesis

This thesis presents the first steps of a larger research initiative to investigate the problem of how to perform computer vision based navigation for spacecraft proximity operations in an unknown environment.

Chapter 2 presents the first step of this research, which is to design and develop a visual navigation testbed, the SPHERES Goggles, which can be used to experimentally evaluate different computer vision based guidance, navigation and control algorithms. This testbed was designed as a hardware upgrade to the SPHERES satellites. It includes two cameras, two illuminating LED lights, a 1 GHz embedded CPU running real-time linux, an 802.11g wireless networking and a 27.5 Watt-hour battery with the associated power electronics. This upgrade is integrated in a mechanical package that has a mass of 895 grams (including the battery). The design of this hardware is intended to be a "flight-traceable" system that can be easily transitioned to operations on the inside of the International Space Station. The details of the hardware trade studies and the design process for the software, hardware and mechanical subsystems are presented in Chapter 2.

Chapter 3 presents the first vision based navigation algorithm that was implemented on the SPHERES Goggles. This system uses fiducial markers to estimate the relative position, orientation, linear and angular velocities between two spacecraft.

An iterative "globally convergent" photogrammetric solution to the exterior orientation problem was used. The exterior orientation stage is followed by a multiplicative Extended Kalman Filter. This filter incorporates the relative dynamics into the estimates, and rejects outlier measurements based on the Mahalanobis distance of the innovation. Experimental results are gathered using the SPHERES Goggles and an experimental upper-bound on the accuracy and precision of all of the state estimates is determined.

Chapter 4 presents the results of a preliminary study to investigate the problems of achieving high performance embedded computing for vision based navigation within an unknown environment. Evidence is presented that strongly suggests that there is a mismatch between the current algorithms that are used for vision based navigation in an unknown environment and the hardware that is used to run these algorithms. Characteristics of typical algorithms, such as inherent data parallelisms, memory access patterns and instruction complexity, are specifically discussed. Additionally, the implications of these characteristics on processing hardware architectures are also described. Using these algorithmic characteristics and the demands they make on hardware architectures, different processors are compared in terms of the expected realistic speed of computation that could be provided per unit of electrical power. This analysis concludes that Graphics Processing Units are the best suited, commercially available, processing architecture for high performance embedded computing for visual navigation in an unknown environment. This chapter concludes with a proposed implementation of a GPU accelerated, dense 3D occupancy grid stereo SLAM algorithm that could solve the problem of visual navigation within an unknown environment in a power efficient manner.

5.2 List of Contributions

The following is a list of the contributions of this thesis.

- Design and development of a flight traceable upgrade to the SPHERES Satellites that enables computer vision based navigation research.

- Implementation and experimental evaluation of an vision based spacecraft relative navigation algorithm that uses a fiducial marker with a known geometry.
- Identification of an approach for the estimation of relative position, orientation, linear and angular velocity using a coplanar fiducial target that is robust to real-world noise and error when using only four (the theoretically minimum number) of coplanar point correspondences.
- Comparison of different processing hardware in terms of characteristics that are relevant to embedded visual navigation systems (i.e. real world giga-Floating Point Operations per Second per Watt).
- The conclusion that currently available commercial GPUs could provide almost 35 times more computational capability than commonly used commercial CPUs for the same electrical power and that it is likely that all of their floating point units can be fully utilized in visual navigation applications within an unknown environment.

5.3 Future Work

The main focus of future work on the SPHERES Goggles will involve the development of the next version of the hardware and software, which would be able to operate inside the International Space Station. The primary issues that are faced are thermal complications due to the lack of convective heat transfer, the relatively low processing capabilities of the Pico-ITX (limited primarily by the limited on-chip cache), and the bandwidth limitations of the current 802.11g wireless network. Operating inside the ISS will introduce a number of additional constraints in terms of safety (lights and batteries), wireless interference and compatibility and operational logistics.

Future work on the fiducial marker relative navigation system should include the incorporation of the inertial measurement units, and thruster actuation commands into the Extended Kalman Filter to further improve the results. Different sizes and styles of targets should be compared in terms of navigational performance. The image

processing components of the algorithm could likely be rewritten to require less time for computation and allow the algorithm to run at a higher frame rate.

A number of areas of future work exist for the preliminary results presented in Chapter 4. An experimental analysis of the bottlenecks of different vision algorithms could be performed. This could lead to a more detailed evaluation of the characteristics of multiple vision based navigation algorithms (not just stereo depth mapping) and their associated hardware requirements. Additionally, the development of a GPU accelerated visual SLAM algorithm would be a very interesting study. This would require an incremental approach to the development of parallelized navigation algorithms along with an experimental evaluation of the system on real world data.

Appendix A

SPHERES-Goggles Design Documentation

A.1 Goggles-SPHERES Software Interface

The Goggles-SPHERES software interface has two main components, the software that resides on the SPHERES satellite (a .img file) and an executable program that runs in linux on the GOGGLES processor. These two processors are connected by a 115.2 kbps RS232 interface. This appendix describes the software interface between these two programs, specifically the code that is residing on the GOGGLES. A useful document is the SPHERES Guest Scientist Program document spheres-gsp.pdf, which can be found online [3].

A.1.1 SPHERES LIIVe Images

Four SPHERES images have been created for the LIIVe project, however they are all slight variations on the same gsp.c file. The main differences have to do with two things:

- Whether filtered or unfiltered gyro data is sent from the SPHERES to the GOGGLES
- Whether the mixer and controller gains take into account the different mass

and inertia properties created by the goggles (the "reconfiguration mixer" image incorporates the mass changes).

Table A.1: SPHERES Boot Images

Directory	Description
<INSTALL_PATH> /liive/sphere1	Base image with filtered gyros
<INSTALL_PATH> /liive/sphere1_unfilter_gyro	Base image with unfiltered gyros
<INSTALL_PATH> /liive_reconfig/sphere1_unfilter_gyro	Base image with filtered gyros, and reconfiguration mixer
<INSTALL_PATH> /liive_reconfig/sphere1_unfilter_gyro	Base image with unfiltered gyros and reconfiguration mixer

The base image has 5 different tests on them. Test 1 is the normal test for general SPHERES operations using global metrology. Tests 2 through 5 have the global estimator disabled and are able to run the controller at a higher frequency.

Table A.2: Base Image Tests

Test Number	Global Metrology Update Period	Control Period	Allowable Control Mode
1	200 ms	1000 ms	TARGET, INERTIAL, BODY
2	Not Available	500 ms	BODY
3	Not Available	200 ms	BODY
4	Not Available	100 ms	BODY
5	Not Available	50 ms	BODY

A.1.2 GOGGLES spheres.h

The GOGGLES API that runs on the Pico-ITX and interfaces with SPHERES is defined in the spheres.h file, and its code is in spheres.c (these files require faked_integrated_accels.h and faked_integrated_accels.c, which are discussed later). The source code for this header file is listed in Appendix B.4.

Useful global variables in spheres.h

A number of global variables are defined that represent the state of the system. These are always overwritten with the latest data from the SPHERES satellite (timestamps are included and described later in this document). It is recommended that the following POSIX standard mutex be used to access this data:

```
1 pthread_mutex_lock(&spheres_data_mutex);
2 position_x = global_data[0];
3 pthread_mutex_unlock(&spheres_data_mutex);
```

float imu_data[3]: This is the rotation rate in radians per second. It is the result of low pass filtering 50 1ms interval samples (the source code for the filter is in pads_convert.c). These values are scaled by a default scale factor and have a bias removed (these have not been calibrated for the individual gyro). The 0, 1 and 2 elements in the array correspond to the X, Y and Z gyro respectively.

float imu_data_unfiltered[3]: This data is exactly the same as above, except that rather than filtering 50 samples on SPHERES and sending the result to the GOGGLES, the latest sample is sent unfiltered.

float global_data[13]: This data is the SPHERES state that is estimated using the global metrology estimator, which takes into account measurements from the ultrasonic beacons and the gyroscopes. Section 5.3 of the spheres-gsp.pdf document defines the elements of the array. Their global coordinate frame is determined by the location of the beacons that are input through the KC GUI program.

int test_number: This value represents the test number that is selected in the KC GUI program (i.e. pressing key "3" to start the test will set this value to 3).

Time Stamps and Synchronization

The SPHERES and GOGGLES processors have two different clocks. The SPHERES clock (measured in ms) begins at 0 when user starts a test in the KC GUI. The GOGGLES clock is the linux CLOCK_REALTIME that is defined in <time.h> (this begins at Jan 1, 1970), and is accessed using the code:

```
1 struct timespec current_time;
2 clock_gettime(CLOCK_REALTIME, &current_time);
```

There are 3 important variables for time synchronization:

- struct timespec start_sys_time
- struct timespec latest_sys_time
- int test_time (number of milliseconds since start of test)

The SPHERES-GOGGLES software interface uses a basic time synchronization mechanism. When the GOGGLES receives a start of test message, it records the current linux time in the start_sys_time variable. Whenever the GOGGLES receives a new gyro or global measurement, it contains a time-stamp of the current SPHERES test time. This is recorded in the test_time variable as well as the current linux time (CLOCK_REALTIME) is recorded in the latest_sys_time. Since the gyro measurements come in every 50 ms, this synchronization occurs every 50 ms and will prevent clock drift over long periods of time.

Therefore if a program running on the goggles wanted to know, exactly what the test time was on SPHERES, it should get the current linux time, subtract this from the latest_sys_time, convert to milliseconds and add the test_time to this.

Functions in spheres.h

The spheres.h file defines a number of functions that will be discussed below. An example program that demonstrates the use of all of these functions is found in goggles.c.

Messages are passed between the SPHERES and GOGGLES using an ASCII protocol. This allows the messages to be inspected by a terminal program such as TeraTerm or minicom. The send functions use sprintf() while the receiving function uses sscanf(). (NOTE: the use of ascii messages will likely change in future releases due to the DSP/BIOS implementation of the SYS_SPRINTF function).

initSpheres() This is the first function that should be called in any program that will interface with SPHERES. The main purpose of this function is to open the serial port with the correct parameters (baud rate, stop bits etc.) and start a new thread (defined in the function spheres_thread(void *ptr)) that is dedicated to processing messages that come from SPHERES. The serial port is opened with the following parameters:

- 115200 baud
- 8 bit
- NO PARITY
- NO STOP BIT

The port is opened in canonical mode (read blocks until a null character is received), so that the thread will automatically block in its "infinite loop". When a new string is received, the function parseString(char* buffer) is called to process the data.

parseString(char* buffer) This function is called automatically by the spheres communications interface thread and does not need to be called by the user. When the thread receives a new string it is parsed using this function. The formats are defined below:

Table A.3: ASCII Data Communication Protocol SPHERES− >GOGGLES

Message	Format	Frequency	Data
Start of Test	ST: %d\r\n	Once per test	Test Number
End of Test	PPPP\r\n	Once per test	
Filtered Gyro	Y[%d]%X,%X,%X\r\n	20 Hz	X, Y, Z stored in imu_data[3]
Unfiltered Gyro	YS[%d]%X,%X,%X\r\n	20 Hz	X, Y, Z stored in imu_data_unfiltered[3]
Global Data	G[%d]%x,%x,%x:%x,%x,%x:%x, %x,%x,%x:%x,%x,%x\r\n	5 or 0 Hz	state stored in global_data

In the gyro and global data, the SPHERES test time is included in the square brackets [%d].

waitGspInitTest() This function provides a mechanism for waiting for the test to start. It will block until a start of test message is received (return 0), or until a 10 second timer elapses (return 1). A start of test message is transmitted by the SPHERES software whenever the user starts a test in the KC GUI (pushing a key from 1 to 9).

checkTestTerminate() In order to detect whether the user has pressed F4 on the KC GUI to stop the test, the GOGGLES software must call this function. A value of 0 is returned from checkTestTerminate() if the test is currently running. However if the test has stopped a 1 is returned. Additionally, this function implements a "software watchdog". If a test has been running, but it has been more than 10 seconds since a message has been received from SPHERES, this function also returns 1.

unsigned int sendCtrl(float* target, int ctrlmode) The sendCtrl function has three different control modes that are #defined in the spheres.h file. These modes are identical in behaviour to their corresponding modes. Each mode accepts a pointer to a float* target array (no out-of-bounds error checking is provided on the target vector).

CTRL_MODE_TARGET: This mode uses a PID controller to regulate the state. The desired state is an array of 13 vectors that is identical in sequence and units (all are m, m/s or rad/s) to the above mentioned Global Data array (e.g. the target[0] element is the X position, target[12] is the Z angular velocity, etc.). If a sequence of waypoints is desired, the GOGGLES code should call sendCtrl with different targets at different times.

CTRL_MODE_INERTIAL: This control mode provides direct control over the forces and torques generated by the SPHERES. The sequence of the target array for elements 0,1,2 are the forces in the X,Y,Z direction respectively and are measured in Newtons. The sequency of the target array for elements 3,4,5 are the torques about

the X,Y and Z axis respectively and are measured in Newton-meters. The reference frame for these forces and torques is in the global/inertial frame that is defined by the beacon's locations.

CTRL_MODE_BODY: This control mode provides direct control over the forces and torques generated by the SPHERES. It is identical to the inertial control mode except that the forces are in the SPHERES body frame (see spheres-gsp.pdf). This is the only control mode that will work in tests 2 through 5 in the liive boot images.

When the sendCtrl function is called, the GOGGLES will immediately transmit a message to the SPHERES controller with the provided parameters. The SPHERES software will receive this message and store the data in a buffer that will be used by the next control iteration. If the SPHERES satellite receives new data before the old data is used, the old data will be over-written. If the GOGGLES stop transmitting control messages, the SPHERES software will use the last received data in the buffer. Therefore no longer calling sendCtrl WILL NOT stop SPHERES from firing its thrusters.

A.1.3 Faked Inertial Accelerometers

One of the requirements of the LIIVe project is the ability to accurately simulate double integrated accelerometers. To do this the SPHERES-GOGGLES API includes the files "faked_integrated_accels.h" and "faked_integrated_accels.c". The spheres.h file includes a state vector **float FIA_global_data[13]**. This state vector is identical to the global_data state vector, however it has errors added to its linear position and linear velocity. This error consists of integrated Gaussian White Noise (generated by the GNU Scientific Library) and a small accelerometer bias that does not drift with time.

The accelerometers that are normally used in SPHERES have bias and noise data that is provided by the manufacturer. Based on this data it was determined that the bias can typically be calibrated to within 0.5 mg (0.0059 m/s^2). Also, at a frequency of 1kHz the SPHERES accelerometer data sheet states that the noise is within 1500 ug-rms, which implies the noise has a standard deviation of 0.0147 m/s^2).

FIA Software API

A small number of functions are used to define this API.

init_FIA_defaults(): This function initializes the random number generator with a random seed based off of linux's "entropy generator" in /dev/urandom. It also sets the default biases and standard deviations.

reset_FIA_errors(): This function sets the integrator errors to zero.

close_FIA(): Frees the memory used by the random number generator.

FIA_timestep(double delta_T): This function integrates the error in discrete time. Since this function is already called by the "parseString(char buffer)" function, it should not normally need to be called by the user. This function integrates based on the following rules:

$$E[w_k] = 0 \tag{A.1}$$

$$E[w_i w_j^T] = \sigma^2 \delta(i-j) \tag{A.2}$$

$$\dot{x}_k = \dot{x}_{k-1} + \Delta T(bias + w_k) \tag{A.3}$$

$$x_k = x_{k-1} + \Delta T(\dot{x}_{k-1}) \tag{A.4}$$

A.2 Goggles Electronics Schematics

In this section the schematics for the Goggles electronics are shown along with the layouts and photos of the populated and assembled boards.

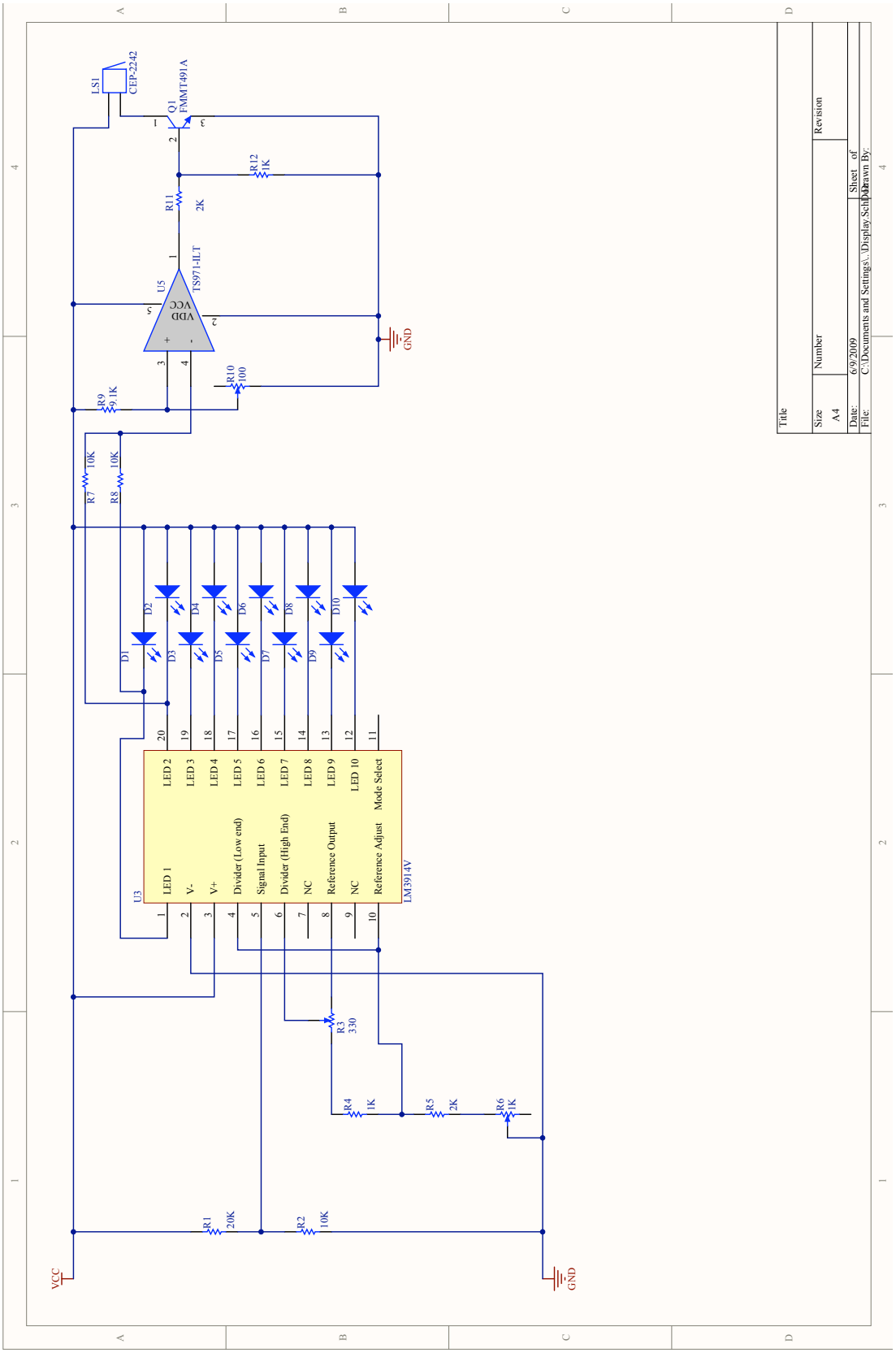


Figure A-1: Combined PCB: Voltage Display

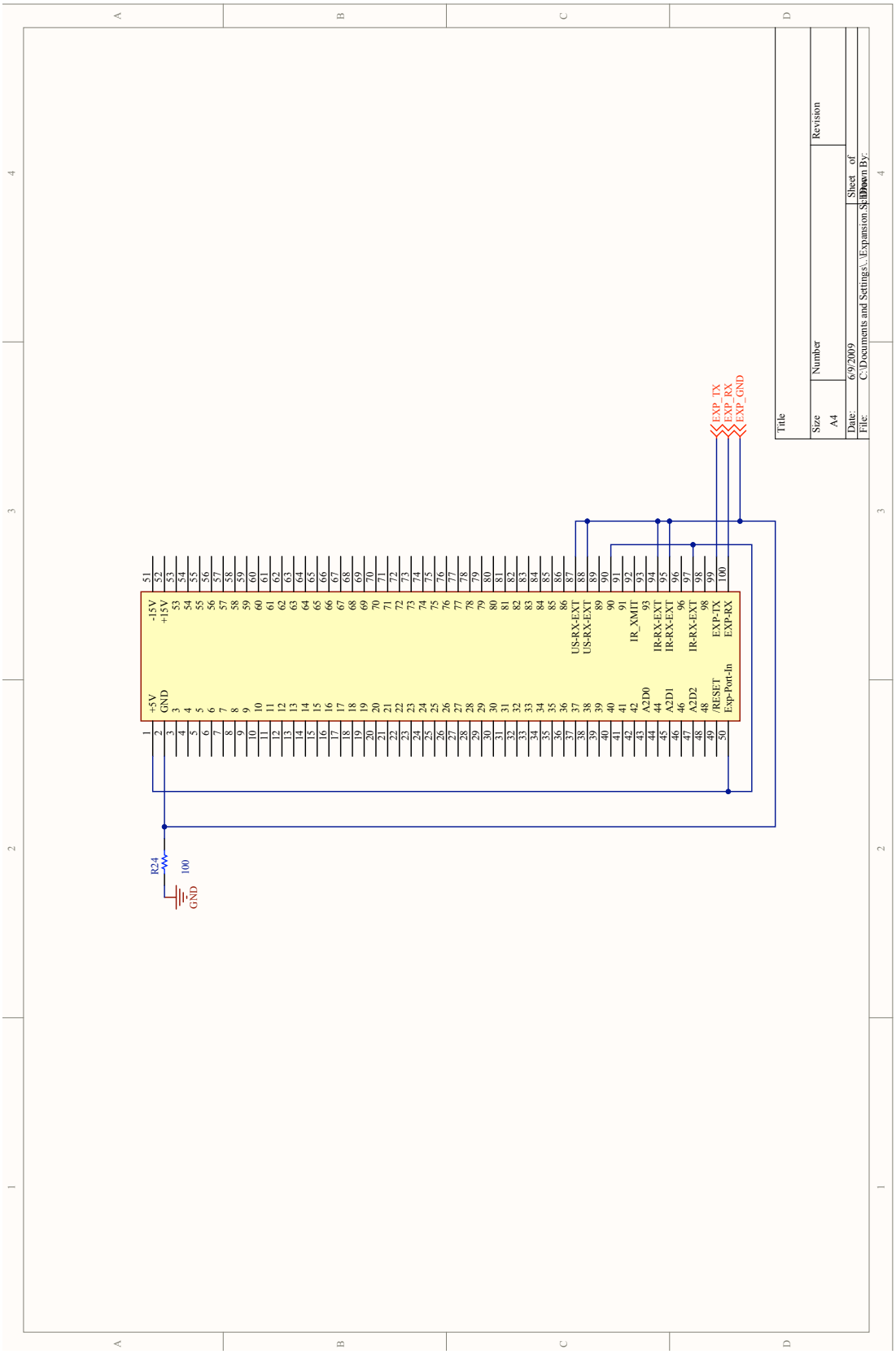


Figure A-2: Combined PCB: Expansion Port Connector

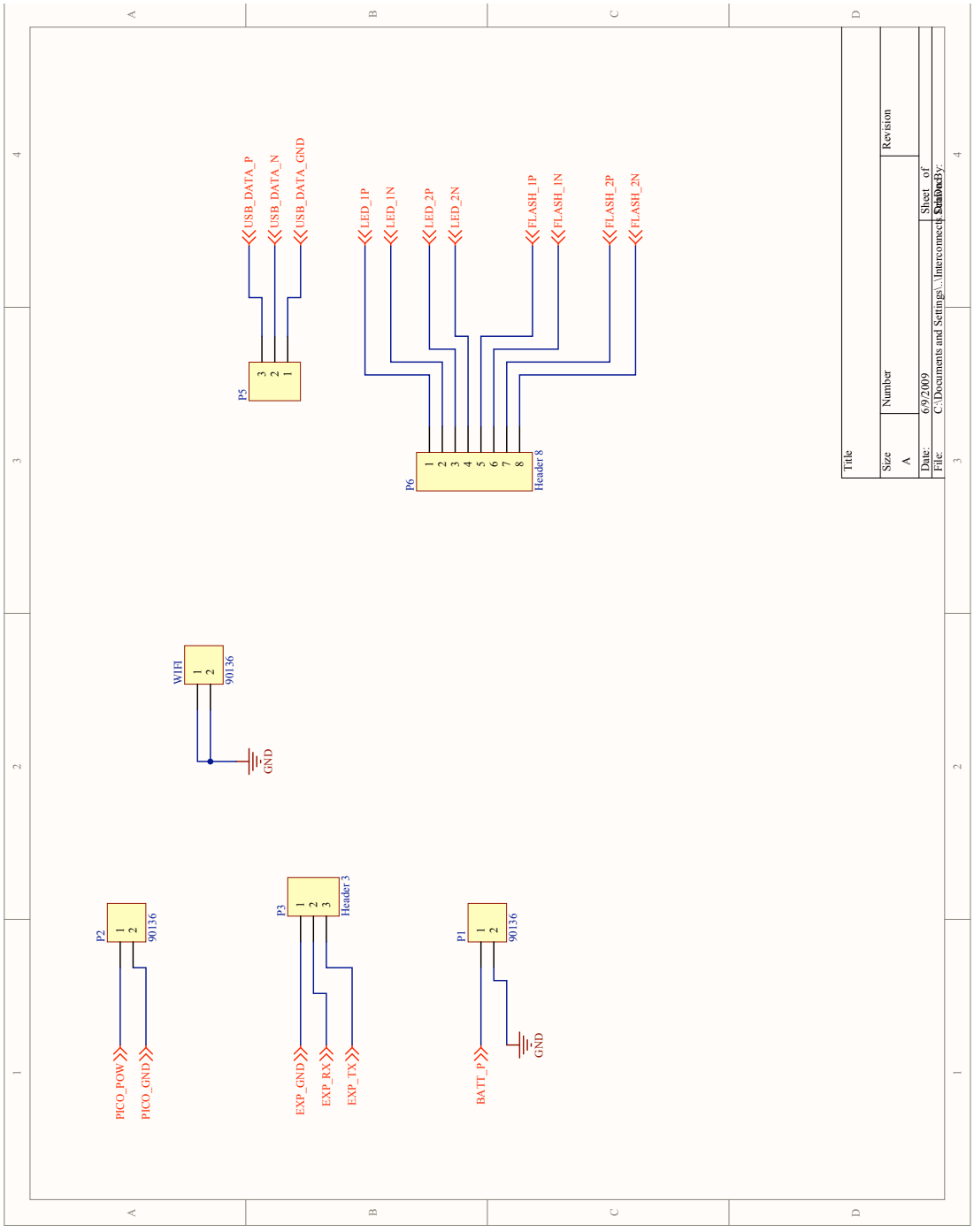


Figure A-3: Combined PCB: Internal Connectors

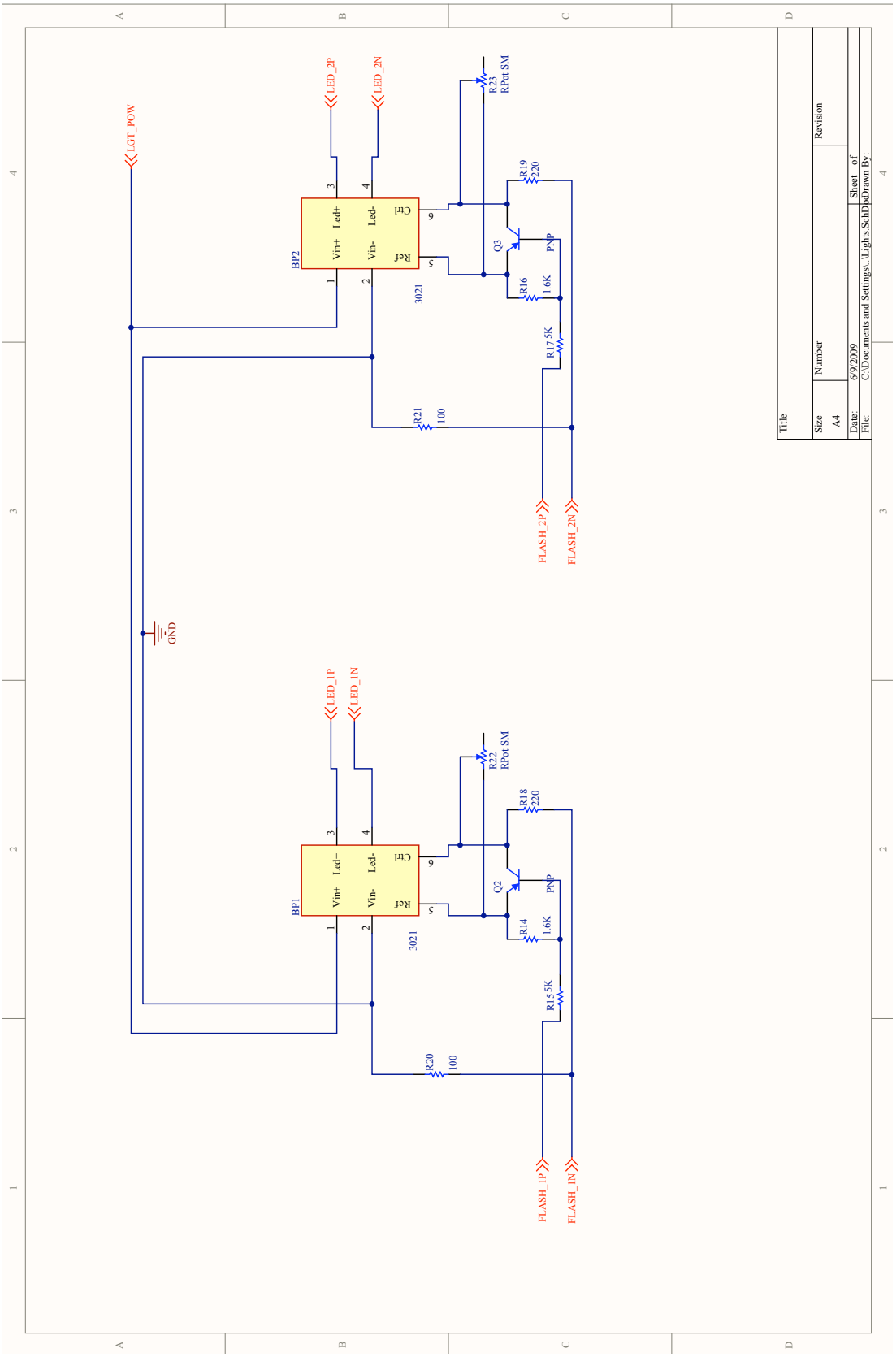


Figure A-4: Combined PCB

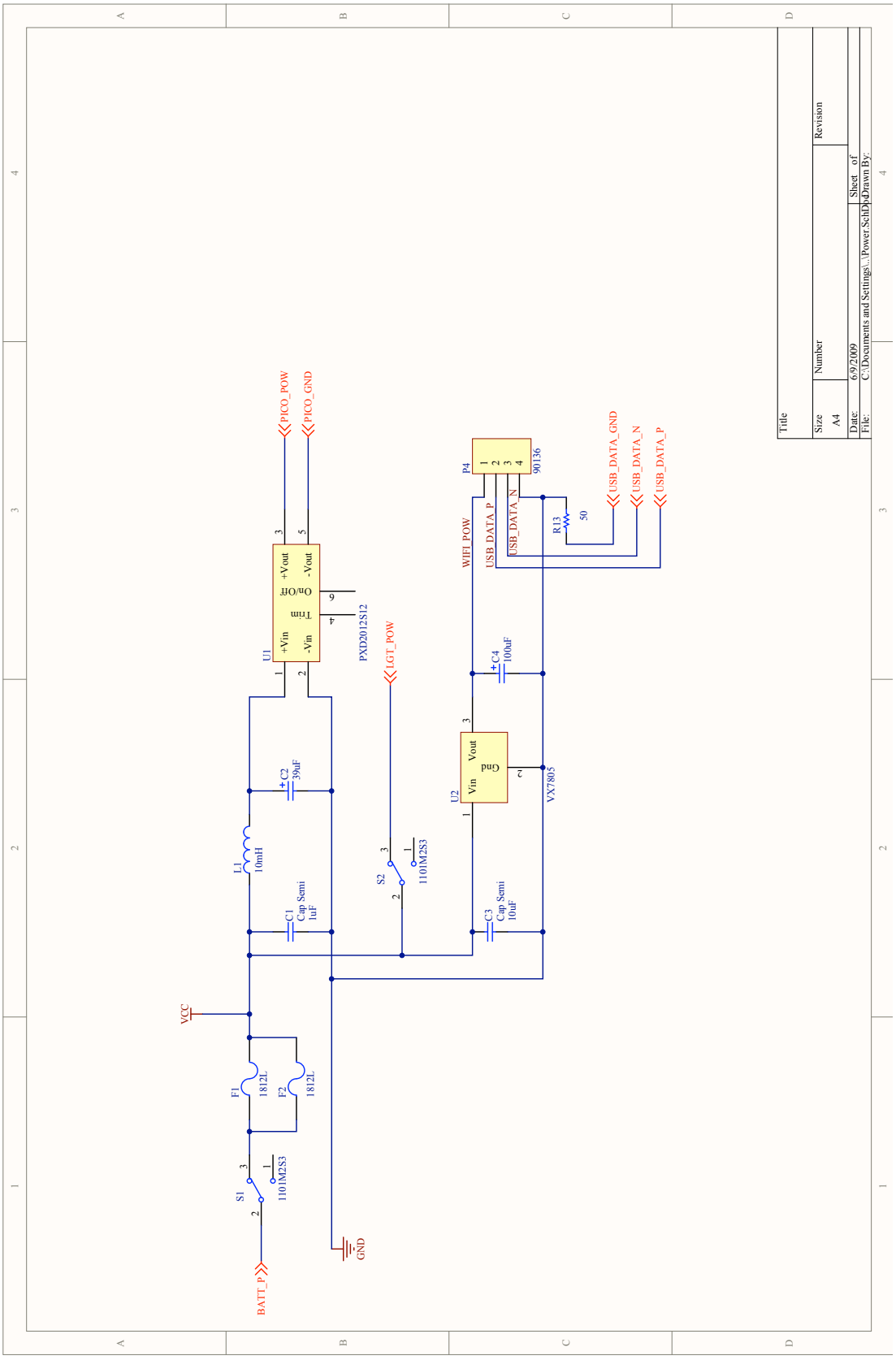


Figure A-5: Combined PCB: Switches and DC-DC Convertors

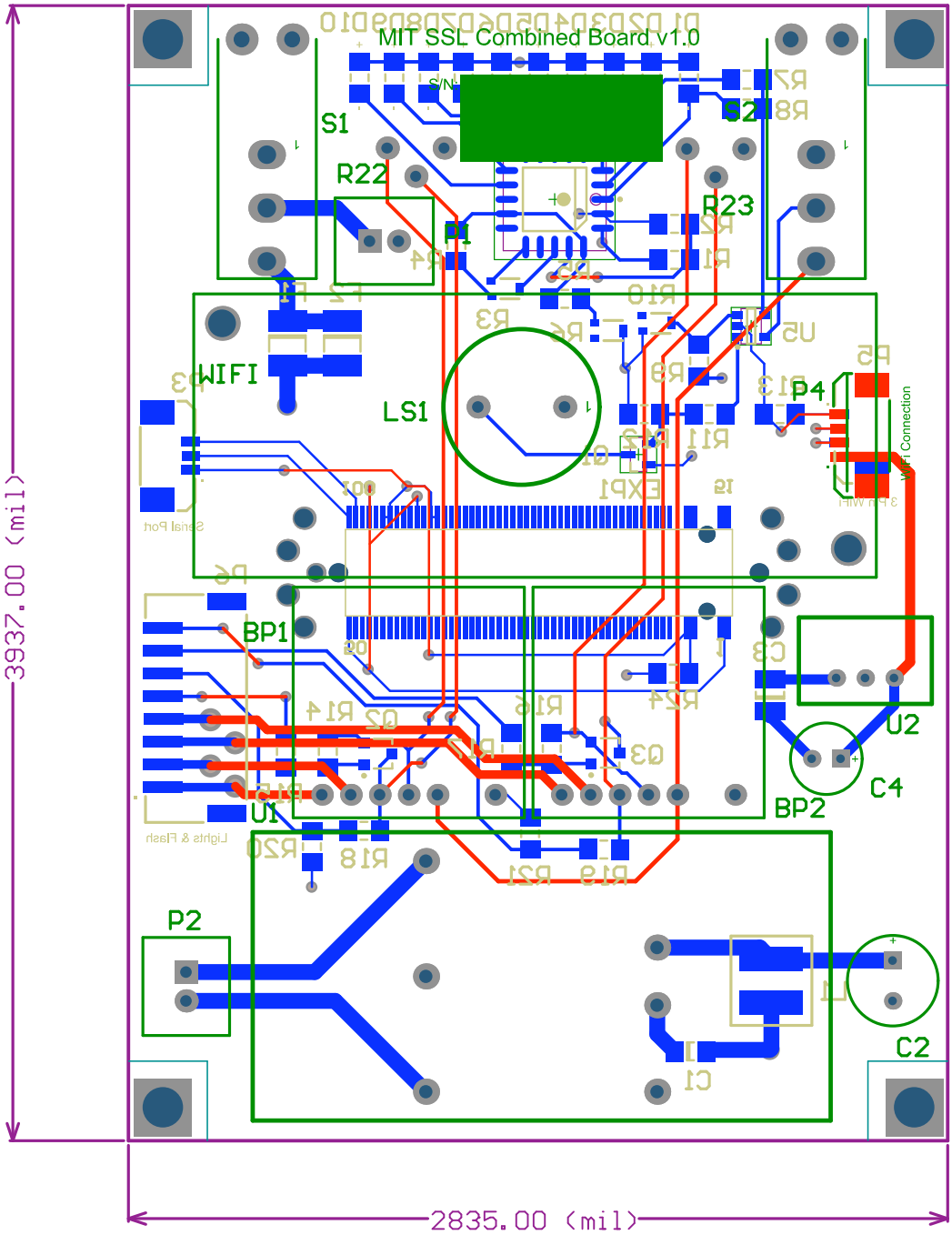


Figure A-6: Combined PCB: Board Layout

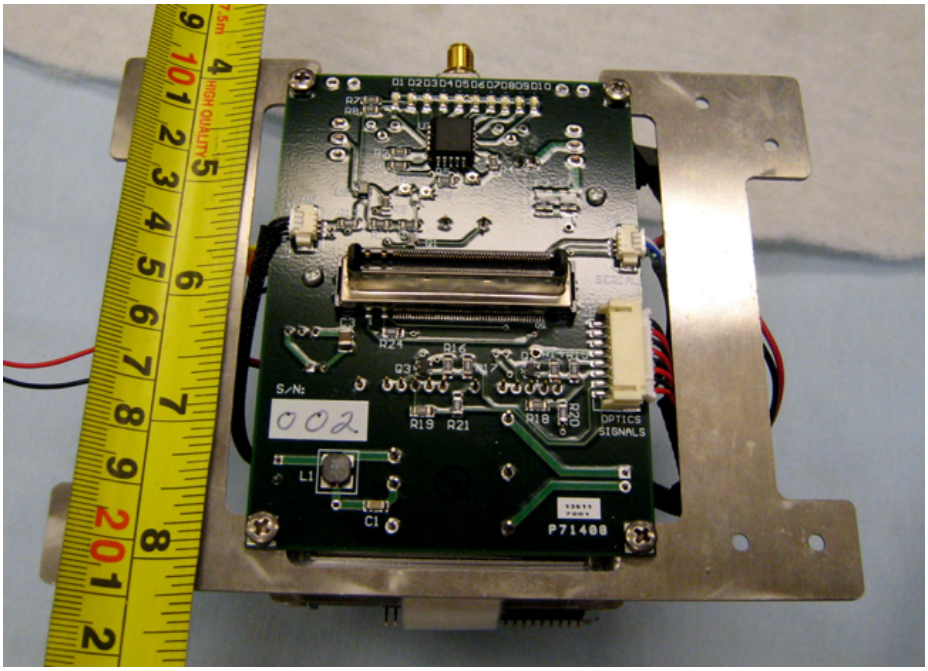


Figure A-7: Combined PCB: Back Photo

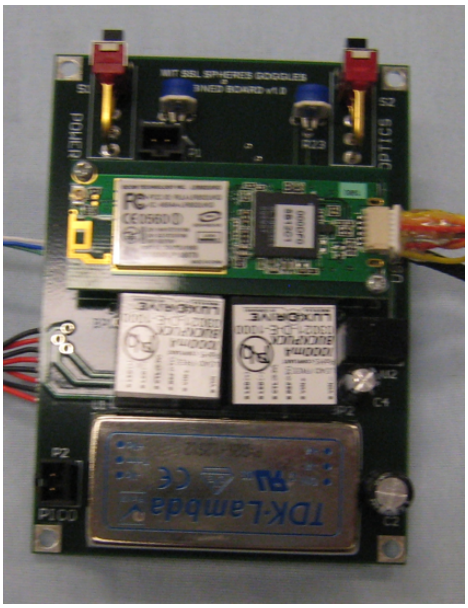


Figure A-8: Combined PCB: Front Photo

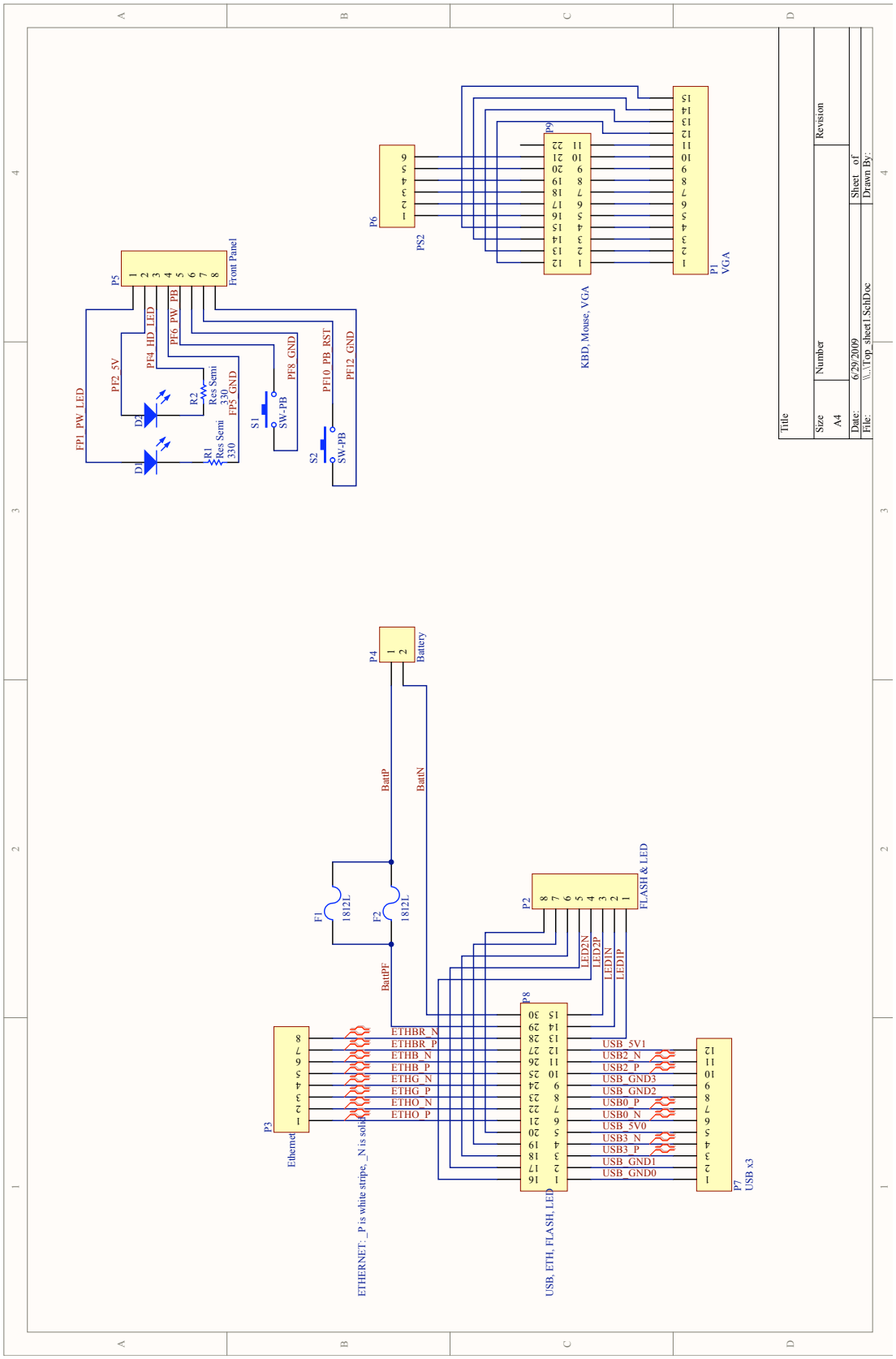


Figure A-9: Top PCB: Schematic

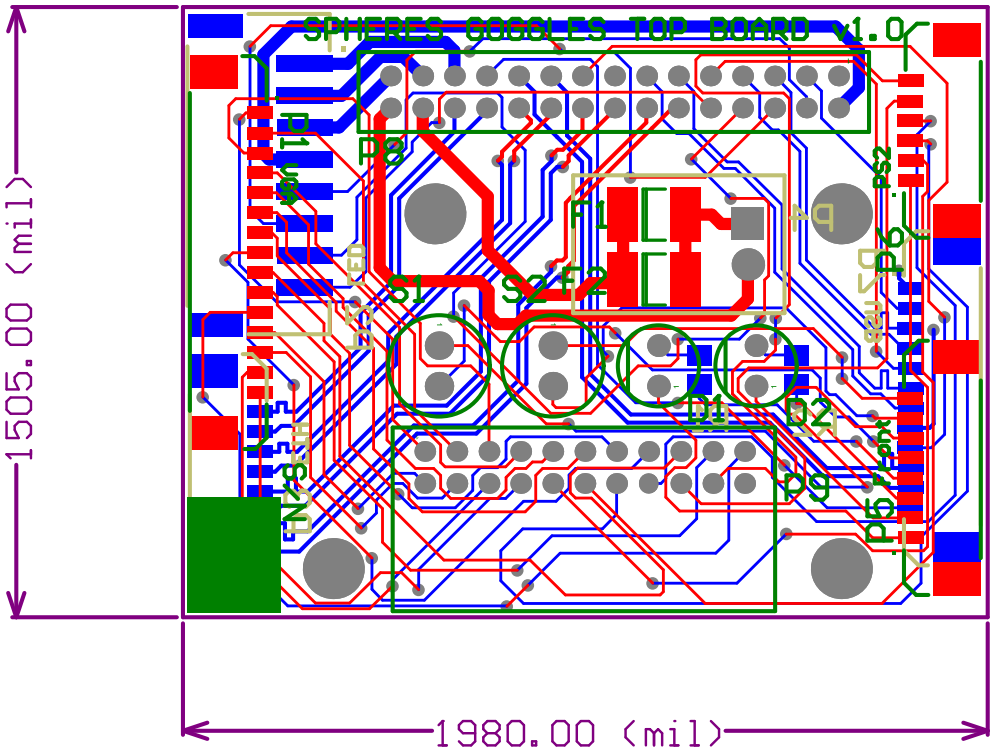


Figure A-10: Top PCB: Board Layout

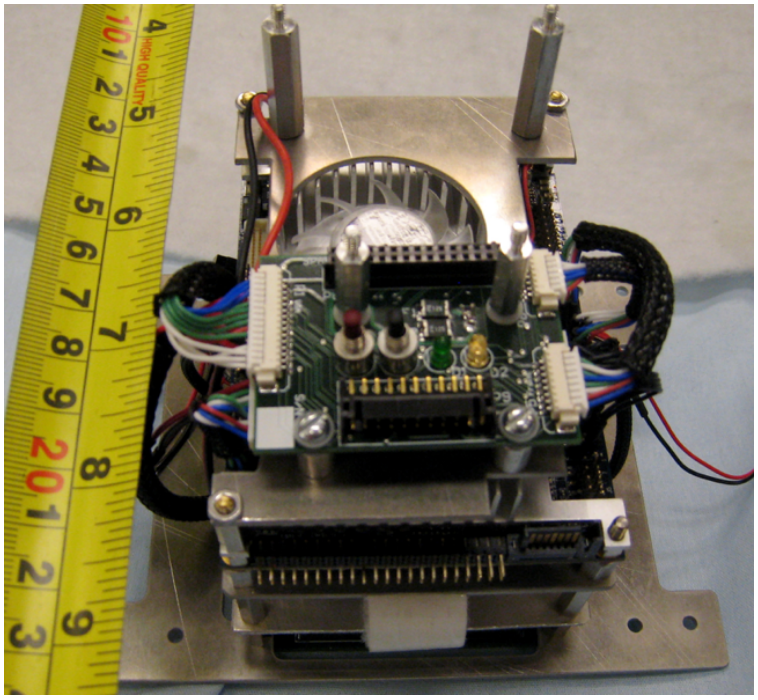


Figure A-11: Top PCB: Front Photo

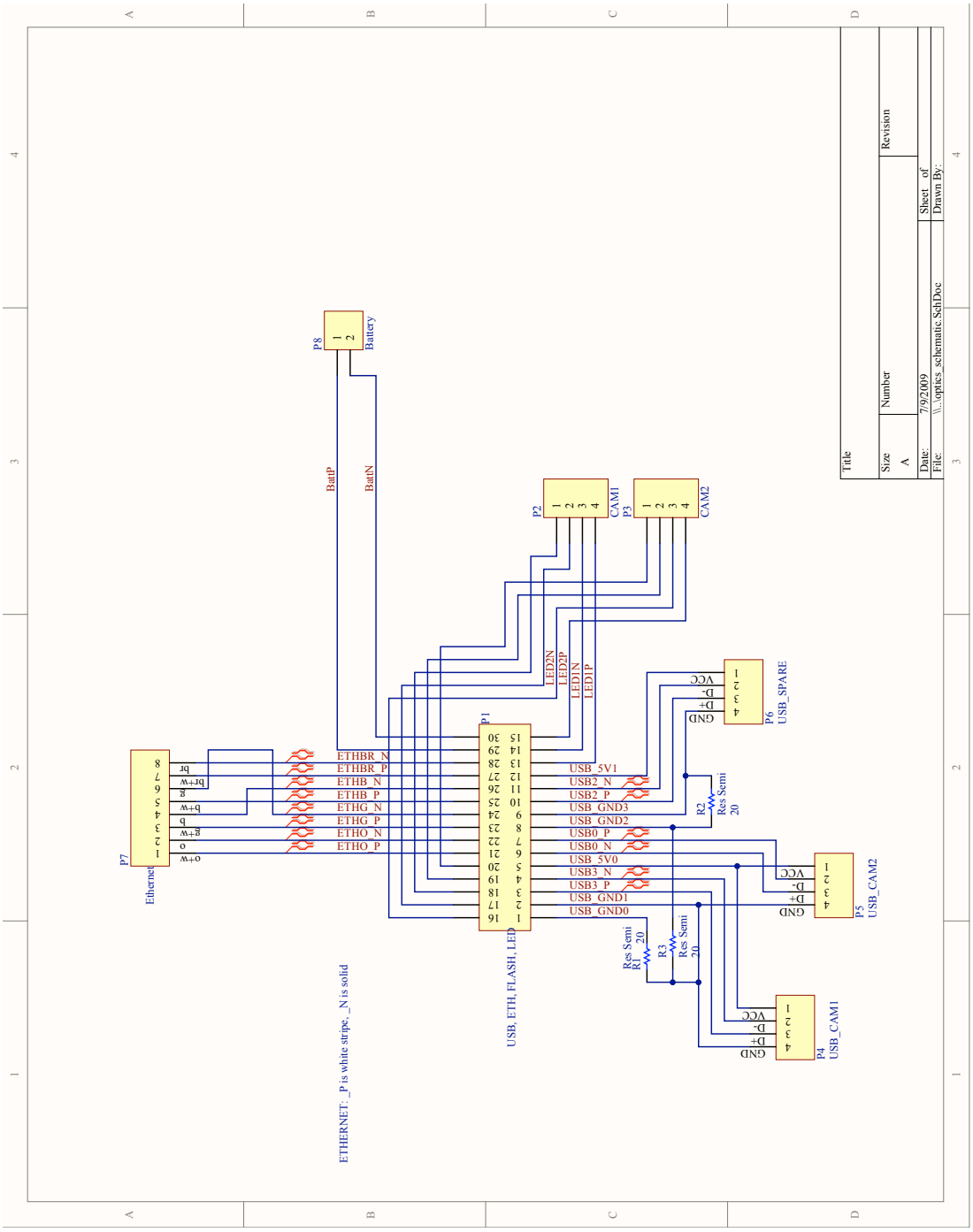


Figure A-12: Optics PCB: Schematic

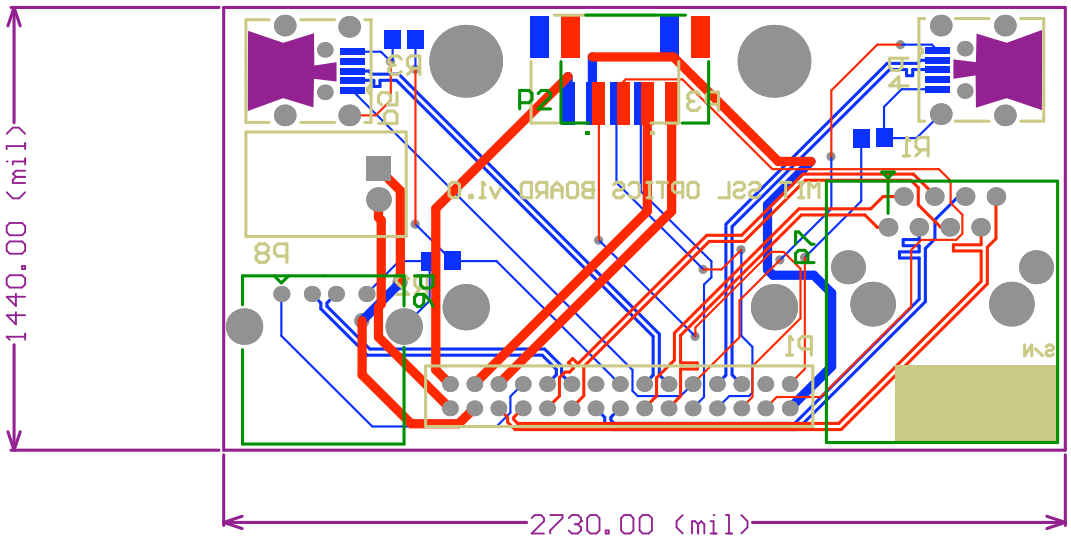


Figure A-13: Optics PCB: Board Layout

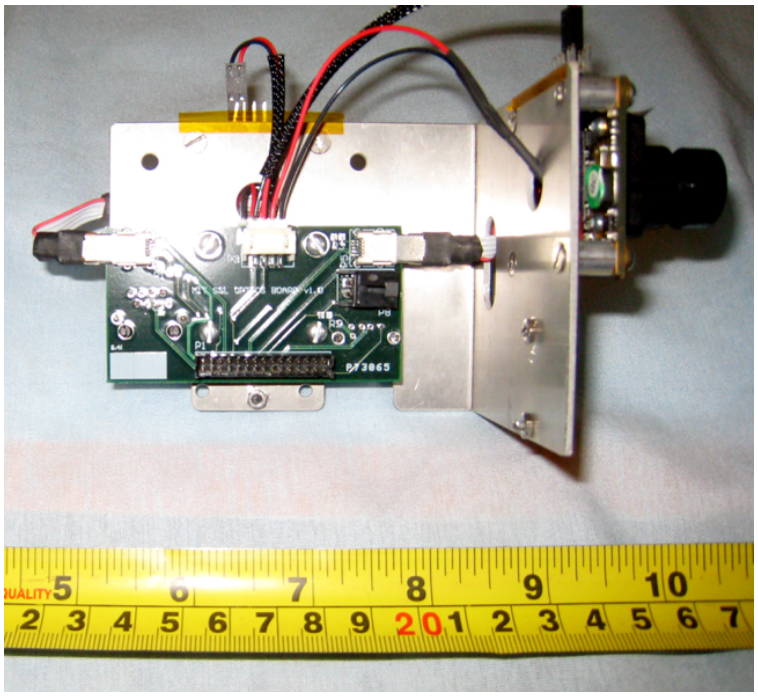


Figure A-14: Optics PCB: Back Photo

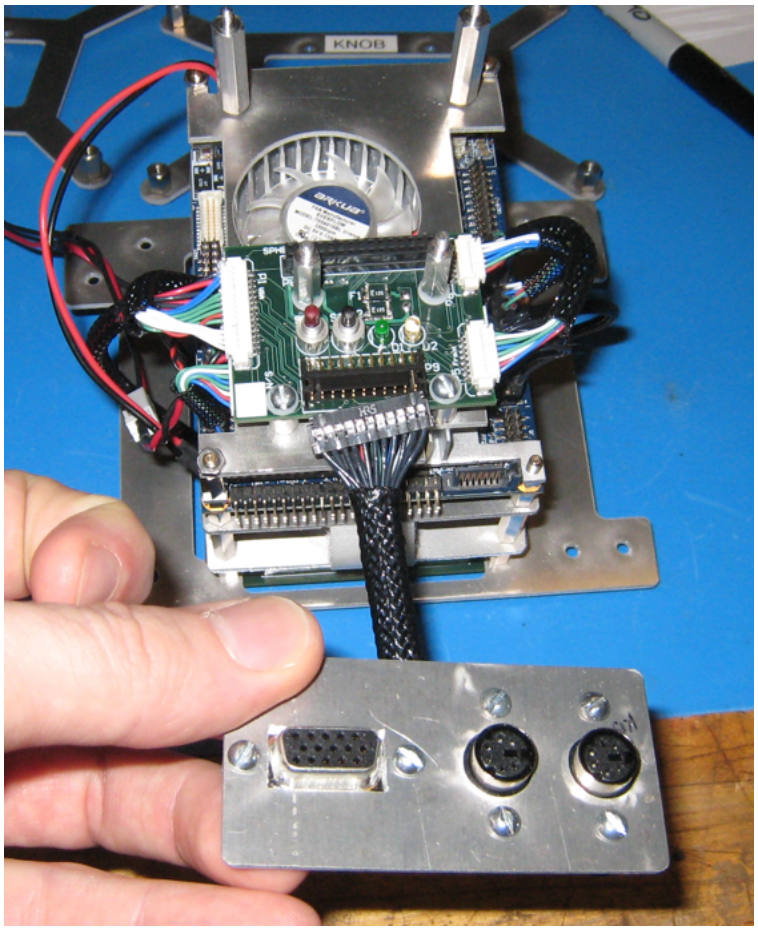


Figure A-15: Optics PCB with VGA, Keyboard and Mouse Connector

Appendix B

Source Code

This appendix contains source code that was used in the experimental implementation of the algorithms. The code was written in Matlab and then converted to C using the Matlab Real-Time Workshop[67].

B.1 Exterior Orientation

The source code here implements the exterior orientation problem described in Section 3.5. It is supported by a function *DCM2quat* which converts from a direction cosine matrix to a quaternion representation.

```
1 function [T, R, q, dk1, dk2, dk3, dk4, iter, rms] = ...
2     haralick_exterior_iter_nonlin(u1, v1, u2, v2, u3, v3, u4, v4, ...
3     x1, y1, x2, y2, x3, y3, x4, y4, d1, d2, d3, d4, f)
4 % ***************************************************************
5 % This function implements the exterior orientation. Its inputs are:
6 %   [u1, v1] ... [u4, v4] - the camera pixel locations of the target
7 %    which correspond to the respect real world targets [x1, y1] ...
8 %        [x4, y4].
9 %   d1 ... d4 is an initial guess for the lengths of the sides of the
10 %       pyramid
11 %   f is the focal lenght in pixels
12 % The outputs are:
13 %   T, R - the translation and rotation matricies
14 %   dk1 ... dk4 - the final values for the lengths of the sides of
```

```
15 %       the pyramid
16 %   iter — the number of iterations of the algorithm
17 %   rms — the mean squared error of the re—projection cost function
18 % ****************************************************************
19
20 %set up initial conditions
21 V1 = [u1/f; v1/f; 1];
22 V2 = [u2/f; v2/f; 1];
23 V3 = [u3/f; v3/f; 1];
24 V4 = [u4/f; v4/f; 1];
25
26 Y1 = [x1; y1; 0];
27 Y2 = [x2; y2; 0];
28 Y3 = [x3; y3; 0];
29 Y4 = [x4; y4; 0];
30
31 dk1 = d1;
32 dk2 = d2;
33 dk3 = d3;
34 dk4 = d4;
35
36 %maximum number of iterations
37 max_iter = 1000;
38 err = 0;
39 derr = 0;
40
41 %apply first absolute orientation
42 [T, R] = haralick_3d(Y1, Y2, Y3, Y4, dk1*V1, dk2*V2, dk3*V3, dk4*V4);
43
44 for iter = 1:max_iter
45
46     %make corrections to lenghts
47     dk1 = (R*Y1+T)'*V1 / (V1'*V1);
48     dk2 = (R*Y2+T)'*V2 / (V2'*V2);
49     dk3 = (R*Y3+T)'*V3 / (V3'*V3);
50     dk4 = (R*Y4+T)'*V4 / (V4'*V4);
51
52     %solve absolute orientation problem
53     [T, R] = haralick_3d...
54         (Y1, Y2, Y3, Y4, dk1*V1, dk2*V2, dk3*V3, dk4*V4);
55
56     %recalculate error and check terminating conditions
57     err_prev = err;
58     err = (R*Y1+T—dk1*V1)'*(R*Y1+T—dk1*V1)...
59         + (R*Y2+T—dk2*V2)'*(R*Y2+T—dk2*V2)...
```

```
60          + (R*Y3+T-dk3*V3)'*(R*Y3+T-dk3*V3)...
61          + (R*Y4+T-dk4*V4)'*(R*Y4+T-dk4*V4);
62
63      derr = err_prev - err;
64
65      if (abs(derr) < 5*10^-9)
66          break
67      end
68  end
69
70  %transform results to desired conventions
71  rms = sqrt(err);
72  R=R';
73  q = DCM2quat(R);
74  end
```

```
1  function [q] = DCM2quat(R)
2  %#eml
3  % ********************************************************************
4  % Converts direction cosine rotation matrix to quaternion
5  % Based on approach described by Sidi's Textbook
6  % ********************************************************************
7  a11 = R(1,1);
8  a12 = R(1,2);
9  a13 = R(1,3);
10 a21 = R(2,1);
11 a22 = R(2,2);
12 a23 = R(2,3);
13 a31 = R(3,1);
14 a32 = R(3,2);
15 a33 = R(3,3);
16
17 qa = zeros(4,1);
18 qb = zeros(4,1);
19 qc = zeros(4,1);
20 qd = zeros(4,1);
21
22 qa(4) = 0.5*sqrt(1+a11+a22+a33);
23 qa(1) = 0.25*(a23-a32)/qa(4);
24 qa(2) = 0.25*(a31-a13)/qa(4);
25 qa(3) = 0.25*(a12-a21)/qa(4);
26
27 qb(1) = 0.5*sqrt(1+a11-a22-a33);
```

```
qb(2) = 0.25*(a12+a21)/qb(1);
qb(3) = 0.25*(a13+a31)/qb(1);
qb(4) = 0.25*(a23+a32)/qb(1);

qc(3) = 0.5*sqrt(1-a11-a22+a33);
qc(1) = 0.25*(a13+a31)/qc(3);
qc(2) = 0.25*(a23+a32)/qc(3);
qc(4) = 0.25*(a12-a21)/qc(3);

qd(2) = 0.5*sqrt(1-a11+a22-a33);
qd(1) = 0.25*(a12+a21)/qd(2);
qd(3) = 0.25*(a23+a32)/qd(2);
qd(4) = 0.25*(a31-a13)/qd(2);

if qa(4) > 1e-10
    q = qa;
elseif qb(1) > 1e-10
    q = qb;
elseif qc(3) > 1e-10
    q = qc;
elseif qd(2) > 1e-10
    q = qd;
else
    q = [1;1;1;1];
%     error('Inertial Quaternion could not be computer');
end
end
```

B.2 Absolute Orientation

The source code in this section implements the absolute orientation method described in Section 3.5.1.

```
function [T, R] = haralick_3d(x1, x2, x3, x4, y1, y2, y3, y4)
% ************************************************************************
% Implements the absolute orientation problem as described by Arun
% Inputs are 3x1 vectors for the location of corresponding points
% in two different frames
% ************************************************************************
x_mean = 0.25*(x1+x2+x3+x4);
y_mean = 0.25*(y1+y2+y3+y4);

B = (x1-x_mean)*(y1-y_mean)'+(x2-x_mean)*(y2-y_mean)'+ ...
    (x3-x_mean)*(y3-y_mean)'+(x4-x_mean)*(y4-y_mean)';

[u, d, v] = svd(B);

R = v*u';
if (det(R) < 0)
    %det(R) == -1
    R = v*[1 0 0; 0 1 0; 0 0 -1]*u';
end

T = y_mean - R*x_mean;
end
```

B.3 Multiplicative Extended Kalman Filter

The source code in this section implements the multiplicative Extended Kalman Filter described in Section 3.6.

```
function [x2, P2, Pa] = MEKF(x1, P1, dt, r_obs, q_obs, eta_squared)
% ********************************************************************
% This function describes the Multiplicative Extended Kalman Filter
% as described in this thesis
% Inputs are the mean and covariance from the prior estimate x1, P1
% The measured rotation and translation, q_obs and r_obs respectively
% The accuracy of the exterior orientation solution eta_squared
% ********************************************************************
%initial declarations
x2 = double(zeros(13,1));
P2 = double(zeros(12,12));
K = zeros(12,6);
invmat = zeros(6,6);
con = 0;

%constants for Mahalanobis distance
thresh_pos = -3.0e5;
thresh_att = -3.0e5;

%initial parameters
d=0.016;    %distance between target points
m = 4.16;   %mass in kg
%SPHERES inertia matrix
J = [2.30E-2,   9.9E-5,     -2.95E-4;
    9.9E-5,     2.42E-2,    -2.54E-5;
    -2.95E-4,   -2.54E-5,   2.14E-2];

%process noise covariance
Q = 10*[0.001*0.04*eye(3),   zeros(3);
     zeros(3),      0.01*4.7873E-4*eye(3)]

%measurement noise convariance
R = 0.01*[0.001*eta_squared * eye(3), zeros(3);
        zeros(3),  0.01*4*eta_squared / d^2 .*eye(3)]

%measurement matrix
C = [eye(3), zeros(3,9);
    zeros(3,6), eye(3), zeros(3)];
```

```
39
40 F = zeros(3,1);
41 T = zeros(3,1);
42
43 x = [x1(1:6); [0;0;0]; x1(11:13)];
44 q = x1(7:10);
45
46 %PREDICTION STAGE OF KALMAN FILTER
47 [xa, qa, A, B, G] = f1(x, q, dt, m, J, F, T); %state predict
48 Pa = A*P1*A'+G*Q*G'; %covariance predict
49 Pa = 0.5*(Pa+Pa'); %enforce symmetric
50
51 %UPDATE STAGE OF KALMAN FILTER
52
53 %convert measurement to modified rodrigues parameters
54 dq = qmult1(q_obs, [-1; -1; -1; 1] .* q);
55 ap_obs = 4*dq(1:3)/(1+dq(4));
56 y_obs = [r_obs; ap_obs];
57 innov = (y_obs-C*xa);
58
59 %compute Mahalanobis distance
60 p_xy_att = -(innov(3:6)'*inv(Pa(3:6,3:6) + R(3:6,3:6))*innov(3:6))
61 p_xy_pos = -(innov(1:3)'*inv(Pa(1:3,1:3) + R(1:3,1:3))*innov(1:3))
62
63 %check Mahalanobis threshold
64 if (p_xy_pos > thresh_pos && p_xy_att > thresh_att && eta_squared > 0)
65     %EKF update equations
66     invmat = C*Pa*C'+R;
67     K = Pa*C'*invmat1(invmat);
68     xp = xa + K*innov;
69     %numerically robust form of covariance update
70     P2 = (eye(12)-K*C)*Pa*(eye(12)-K*C)' + K*R*K';
71
72     % Reset quaternion
73     da = xp(7:9);
74     dq2 = [8*da; 16 - da'*da] ./(16-da'*da);
75     dq2 = dq2 / sqrt(dq2'*dq2);
76     q2 = qmult1(dq2, q);
77
78     x2(1:6) = xp(1:6);
79     x2(7:10) = q2;
80     x2(11:13) = xp(10:12);
81 else %just propagate state
82     if eta_squared > 0
83         outlier = 1
```

```
84      end
85      eta = -1;
86      x2(1:6) = xa(1:6);
87      x2(7:10) = qa;
88      x2(11:13) = xa(10:12);
89      P2 = Pa;
90  end
91  P2 = 0.5*(P2'+P2); %make covariance matrix symmetric
92  end
93
94  function [x2, q2, A, B, G] = f1(x, q, dt, m, J, F, T)
95  %computes nonlinear model, described in Chapter 3
96  x2 = zeros(12,1);
97  q2 = zeros(4,1);
98
99  r = x(1:3);
100 v = x(4:6);
101 a = x(7:9);
102 w = x(10:12);
103
104 iJ = inv(J);
105 J11 = J(1,1);
106 J12 = J(1,2);
107 J13 = J(1,3);
108 J21 = J(2,1);
109 J22 = J(2,2);
110 J23 = J(2,3);
111 J31 = J(3,1);
112 J32 = J(3,2);
113 J33 = J(3,3);
114 w1 = w(1);
115 w2 = w(2);
116 w3 = w(3);
117
118 %attitude error
119 w_cross = [ 0,      -w(3),  w(2);
120             w(3),   0,      -w(1);
121             -w(2),  w(1),   0];
122
123 %reference quaternion
124 O = [   0,          w(3),       -w(2),      w(1);
125         -w(3),      0,          w(1),       w(2);
126         w(2),       -w(1),      0,          w(3);
127         -w(1)       -w(2),      -w(3),      0];
128
```

```
129 q2 = expm(0.5*O.*dt)*q;
130 dq = 0.5*O*q
131
132 da1 = 4*(dq(1)*(1+q(4))-dq(4)*q(1))/(1+q(4))^2
133 da2 = 4*(dq(2)*(1+q(4))-dq(4)*q(2))/(1+q(4))^2
134 da3 = 4*(dq(3)*(1+q(4))-dq(4)*q(3))/(1+q(4))^2
135
136 a = [da1; da2; da3].*dt;
137
138 AM = [-w_cross ./2 ,    eye(3),    zeros(3);
139       zeros(3),         zeros(3),       iJ;
140       zeros(3),         zeros(3),       zeros(3)];
141
142 Phi_m = expm(AM.*dt);
143
144 A = [   eye(3),     eye(3).*dt,     zeros(3),               zeros(3);
145         zeros(3),   eye(3),         zeros(3),               zeros(3);
146         zeros(3),   zeros(3),       Phi_m(1:3, 1:6);
147         zeros(3),   zeros(3),       zeros(3),               eye(3)];
148
149 G = [   eye(3)*dt^2/(2*m),  zeros(3);
150         eye(3)*dt/m,        zeros(3);
151         zeros(3),           Phi_m(1:3,7:9);
152         zeros(3),           Phi_m(4:6,7:9)];
153
154 B = G;
155 x2 = A*x;
156 x2(7:9) = a;
157
158 end
159
160 function qout = qmult1(q1, q2)
161 %multiplies two quaternions
162
163 qout = zeros(4,1);
164
165 vq1 = q1(1:3);
166 kq1 = q1(4);
167 vq2 = q2(1:3);
168 kq2 = q2(4);
169
170 q1_cross = [0, -q1(3), q1(2);
171             q1(3), 0, -q1(1);
172             -q1(2), q1(1), 0];
173
```

```
174 qout(1:3) = kq1 .* vq2 + kq2 .* vq1 - q1_cross*vq2;
175 qout(4) = kq1*kq2 - vq1'*vq2;
176 qout = qout/norm(qout);
177 end
178
179 function B = invmat1(A)
180 %inverts matrix in a numerically stable way
181 mat = 0.5*(A+A');
182 [q r] = qr(mat);
183 B = r \ q';
184 end
```

B.4 Goggles Core API Header Files

B.4.1 Header Files

The header files for the Goggles Core API are listed in this section.

```
1 //spheres.h
2 #ifndef __SPHERES__
3 #define __SPHERES__
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <sys/time.h>
8 #include <pthread.h>
9 #include <unistd.h>
10 #include <fcntl.h>
11 #include <termios.h>
12 #include <string.h>
13
14 #include "faked_integrated_accels.h"
15
16 #define SERIAL_PORT "/dev/ttyS0"
17
18 //0 means turn off printfs, 1 means turn on printfs
19 #define VERBOSE_SPHERES  0
20
21 #define CTRL_MODE_TARGET 1
22 #define CTRL_MODE_INERTIAL    2
23 #define CTRL_MODE_BODY        3
```

```
24
25 #define STATE_LENGTH     13
26 #define POS_X       0
27 #define POS_Y       1
28 #define POS_Z       2
29 #define VEL_X       3
30 #define VEL_Y       4
31 #define VEL_Z       5
32 #define QUAT_1      6
33 #define QUAT_2      7
34 #define QUAT_3      8
35 #define QUAT_4      9
36 #define RATE_X      10
37 #define RATE_Y      11
38 #define RATE_Z      12
39
40 #define DEFAULT_GYRO_BIAS     2047.0f        //[counts] of ADC
41 #define DEFAULT_GYRO_SCALE    0.70000e-3f    //[rad/s /count]
42
43 typedef float state_vector[STATE_LENGTH];
44
45 pthread_t comm_thread;
46 int serial_fd;
47
48 int test_running_flag;
49
50 //global data variables -> always contain most recent data
51 extern float imu_data[3];
52 extern float imu_data_unfiltered[3];
53 extern float global_data[13];
54 extern float FIA_global_data[13];
55 extern int test_time;
56 extern int test_number;
57 extern struct timespec latest_sys_time;
58 extern struct timespec start_sys_time;
59
60 pthread_mutex_t spheres_data_mutex;
61
62 unsigned int initSpheres();
63 unsigned int closeSpheres();
64 void *spheres_thread(void *ptr);
65 unsigned int parseString(char* buffer);
66 unsigned int waitGspInitTest();
67 unsigned int checkTestTerminate();
68 unsigned int sendCtrl(float* target, int ctrlmode);
```

```
69 int initNoSpheres();
70
71 #endif
```

```
1 //optics.h
2 #ifndef __OPTICS__
3 #define __OPTICS__
4
5 #define __LINUX__
6 #define IMAGE_BUFFER_COUNT 3
7
8 #include <stdio.h>
9 #include <string.h>
10 #include <uEye.h>
11
12 #define FRAME_RATE  10.0
13 #define EXPOSURE_TIME    25
14 #define FLASH_DELAY 70000
15 #define FLASH_DURATION   30000
16 #define IMG_WIDTH   640
17 #define IMG_HEIGHT  480
18 #define IMG_BITS_PIXEL   8
19
20
21 typedef struct _UEYE_IMAGE
22 {
23     char    *pBuf;
24     INT     img_id;
25     INT     img_seqNum;
26     INT nBufferSize;
27 } UEYE_IMAGE;
28
29 UEYE_IMAGE img_buffer1[IMAGE_BUFFER_COUNT];
30 UEYE_IMAGE img_buffer2[IMAGE_BUFFER_COUNT];
31
32
33 HIDS h_cam1, h_cam2;
34 CAMINFO camInfo1, camInfo2;
35 char * act_img_buf1, * act_img_buf2;
36
37 unsigned int initTwoCameras();
38 unsigned int closeTwoCameras();
39 unsigned int startTwoCameras();
```

```
40 unsigned int stopTwoCameras();
41 unsigned int captureTwoImages(char** last_img_buf1,
42      char** last_img_buf2, int* img_num1, int* img_num2);
43 unsigned int saveTwoFrames(char** frame1,
44      char** frame2, int imgNum1, int imgNum2);
45
46 #endif
```

```
1 //networking.h
2 #ifndef __NETWORKING__
3 #define __NETWORKING__
4
5 #include <unistd.h>
6 #include <sys/socket.h>
7 #include <netinet/in.h>
8 #include <arpa/inet.h>
9 #include <string.h>
10 #include <errno.h>
11
12 #include <stdio.h>
13 #include <stdlib.h>
14
15 #define DEFAULT_SERVER_IP_ADDRESS  "127.0.0.1"    //"18.33.6.155"
16 #define PORT             4000
17 #define COMPRESSION 0
18 #define COMPRESS_LEVEL   1
19
20 //server sockets
21 extern int sockfd;
22 extern struct sockaddr_in servaddr;
23 extern char server_ip[20];
24 extern int lowres;
25
26 int initTCPNetworkSend();
27 int initTCPNetworkReceive();
28 int closeNetwork();
29 int tcpSendImage(int img_num, unsigned char* image_data);
30 int tcpReceiveImage(unsigned char* img_buffer1,
31      unsigned char* img_buffer2);
32 int parseTCPPacket(unsigned char* img_buffer1,
33      unsigned char* img_buffer2);
34 int downscaleImage(unsigned char * bigImage,
35      unsigned char * smallImage);
```

```
36
37 #endif
```

```
1 //faked_integrated_accels.h
2 #ifndef FAKED_INTEGRATED_ACCELS_H
3 #define FAKED_INTEGRATED_ACCELS_H
4
5 #include <stdio.h>
6 #include <gsl/gsl_rng.h>
7 #include <gsl/gsl_randist.h>
8
9 #define FIA_X  0
10 #define FIA_Y  1
11 #define FIA_Z  2
12
13 struct params_FIA {
14      double bias[3];
15      double stddev[3];
16 } params_FIA;
17
18 extern double error_velocity[3];
19 extern double error_position[3];
20
21 void init_FIA_defaults();
22 int reset_FIA_errors();
23 int close_FIA();
24 int FIA_timestep(double Δ_T);
25
26 #endif
```

B.4.2 Example Goggles Program with Networking

This section shows the c code for an example Goggles program that captures images and transfers them over a network connection.

```
1 //goggles_inspector.c
2 /* This function runs onboard the goggles
3 Pico-ITX. It captures images and sends
4 them over a network TCP/IP connection to
5 a program running monitor.c
```

```
6 */
7
8 #include <stdio.h>
9 #include <pthread.h>
10 #include <string.h>
11
12 #include "optics.h"
13 #include "spheres.h"
14 #include "networking.h"
15 #include "faked_integrated_accels.h"
16
17 #define CAMERA_1_SERIAL       "4002718494"
18
19 int nosphere, windowNumber;
20
21 int optproc(int argc, char *argv[]);
22
23 int main(int argc, char *argv[] )
24 {
25      int imgNum1;
26      int imgNum2;
27      char* frame1;
28      char* frame2;
29      int retVal;
30      int flags, ch;
31      int ring = 0;
32
33      state_vector controlSignal;
34
35      optproc(argc, argv);
36
37      retVal = initTwoCameras();
38      if (retVal != 0)
39      {
40           printf("initTwoCameras Failed (Code: %d)\n", retVal);
41           exit(retVal);
42      }
43
44      retVal = initTCPNetworkSend();
45      if (retVal != 0)
46      {
47           exit(retVal);
48      }
49
50      if (nosphere == 0)
```

```
51      {
52           initSpheres();
53
54           retVal = waitGspInitTest();
55           if (retVal != 0)
56           {
57                exit(0);
58           }
59      }
60      else
61      {
62           initNoSpheres();
63      }
64
65      retVal = startTwoCameras();
66      if (retVal != 0)
67      {
68           printf("startTwoCameras Failed (Code: %d)\n", retVal);
69           exit(retVal);
70      }
71
72      //the frequency that this loop is executed is based on the
73      //frequency of the cameras when captureTwoImages is called.
74      //This frequency is defined in optics.h as the FRAME_RATE
75      //define. This is currently set to 10Hz
76      while(1)
77      {
78 //         printf("Ring %d\n", ring);
79
80           retVal = captureTwoImages(&frame1, &frame2,
81                &imgNum1, &imgNum2);
82           if (retVal != 0)
83           {
84                printf("captureTwoImages Failed (Code: %d)\n", retVal);
85                exit(retVal);
86           }
87
88           if (windowNumber == 2)
89           {
90                retVal = tcpSendImage
91                ((strcmp(camInfo1.SerNo, CAMERA_1_SERIAL)) ? 0 : 1, frame1);
92                if (retVal != 0)
93                {
94                     printf("UDP Frame 1 Send Failed
95                     (Code: %d)\n", retVal);
```

```
96                    break;
97               }
98
99               retVal = tcpSendImage
100              ((strcmp(camInfo1.SerNo, CAMERA_1_SERIAL)) ? 1 : 0, frame2);
101              if (retVal != 0)
102              {
103                   printf("UDP Frame 2Send Failed
104                   (Code: %d)\n", retVal);
105                   break;
106              }
107         }
108         else
109         {
110              if (strcmp(camInfo1.SerNo, CAMERA_1_SERIAL))
111              {
112                   retVal = tcpSendImage(0, frame1);
113                   if (retVal != 0)
114                   {
115                        printf("UDP Frame 1 Send Failed
116                             (Code: %d)\n", retVal);
117                        break;
118                   }
119              }
120              else
121              {
122                   retVal = tcpSendImage(0, frame2);
123                   if (retVal != 0)
124                   {
125                        printf("UDP Frame 1 Send Failed
126                             (Code: %d)\n", retVal);
127                        break;
128                   }
129              }
130         }
131         ring++;
132
133         if (nosphere == 0)
134         {
135              retVal = checkTestTerminate();
136              if (retVal != 0) //ESC
137              {
138                   break;
139              }
140         }
```

```
141     }
142
143     retVal = stopTwoCameras();
144     if (retVal != 0)
145     {
146          printf("stopTwoCameras Failed (Code: %d)\n", retVal);
147          exit(retVal);
148     }
149
150     retVal = closeTwoCameras();
151     if (retVal != 0)
152     {
153          printf("closeTwoCameras Failed (Code: %d)\n", retVal);
154          exit(retVal);
155     }
156
157     closeNetwork();
158
159     if (nosphere == 0)
160     {
161          closeSpheres();
162     }
163
164     exit(0);
165 }
166
167 int optproc(int argc, char *argv[])
168 {
169     int i;
170     nosphere = 0;
171     lowres = 0;
172     windowNumber = 2;
173
174     if (argc < 1)
175     {
176          printf("Insufficient arguments\n");
177          return -1;
178     }
179
180     for(i=1; i<argc; i++)
181     {
182          if (strcmp("-nosphere", argv[i]) == 0)
183          {
184               nosphere = 1;
185          }
```

```
186          else if (strcmp("-lowres", argv[i]) == 0)
187          {
188               lowres = 1;
189          }
190          else if (strcmp("-onewindow", argv[i]) == 0)
191          {
192               windowNumber = 1;
193          }
194     }
195
196     return 0;
197 }
```

```
1 //monitor.c
2 /* This program runs on the network and
3 receives images from the program running
4 gogglesInspector.c. This program will
5 display those images using the OpenCV
6 API
7 */
8 #include "monitor.h"
9 #include "networking.h"
10 #include "optics.h"
11
12 IplImage* cvFrame1;
13 IplImage* cvFrame2;
14
15 IplImage* tempFrame;
16
17 int record;
18 CvVideoWriter * vidWriter;
19
20 int initDisplayWindow(int number)
21 {
22     int width, height;
23
24     if (lowres == 0)
25     {
26          width = NET_IMG_WIDTH;
27          height = NET_IMG_HEIGHT;
28     } else
29     {
30          width = 320;
```

```
31          height = 240;
32      }
33      cvFrame1=cvCreateImage(cvSize(width,height),IPL_DEPTH_8U,1);
34      cvNamedWindow("GOGGLES FRAME 1", CV_WINDOW_AUTOSIZE);
35      cvMoveWindow("GOGGLES FRAME 1", 2000,0);
36
37      if (number == 2)
38      {
39          cvFrame2=cvCreateImage(cvSize(width,height),IPL_DEPTH_8U,1);
40          cvNamedWindow("GOGGLES FRAME 2", CV_WINDOW_AUTOSIZE);
41          cvMoveWindow("GOGGLES FRAME 2", 2700,0);
42      }
43
44      return 0;
45 }
46
47 int updateDisplayWindow(int number)
48 {
49      char keystroke;
50
51      if (record == 1)
52      {
53          cvConvertImage(cvFrame1, tempFrame, 0);
54          cvShowImage("GOGGLES FRAME 1", tempFrame);
55          cvWriteFrame(vidWriter, tempFrame);
56      }
57      else
58      {
59          cvShowImage("GOGGLES FRAME 1", cvFrame1);
60      }
61
62      if (number == 2)
63      {
64          cvShowImage("GOGGLES FRAME 2", cvFrame2);
65      }
66
67      keystroke = cvWaitKey(10);
68      if (keystroke == 0x1B)    //ascii ESC
69          return 3;
70      return 0;
71 }
72
73 int closeDisplayWindow(int number)
74 {
75      cvDestroyWindow("GOGGLES FRAME 1");
```

```
76      cvReleaseImage(&cvFrame1);
77      if (number == 2)
78      {
79           cvDestroyWindow("GOGGLES FRAME 2");
80           cvReleaseImage(&cvFrame2);
81      }
82
83      if (record == 1)
84      {
85           cvReleaseVideoWriter(&vidWriter);
86      }
87
88      return 0;
89 }
```

Appendix C

Mathematical Review

This Appendix presents further details and proofs for a number of mathematical concepts that are used in this thesis.

C.1 Parameterizations of Rotation

C.1.1 Euler Angles and Rotation Matricies

The simplest to understand representation of a rotation are the Euler angles. They consist of three angles ψ, θ, ϕ. ψ is a right handed rotation about the z-axis, θ is a right handed rotation about the y-axis, and ψ is a right handed rotation about the x-axis. Since any sequence of rotations *does not commute* in general, a convention for the order of the applied rotations must be specified and adhered to. Additionally, it must be specified whether these rotations take place about the body-fixed axis or about a space fixed axis, as these are not equivalent. In the case of body fixed axis rotations, it is possible for the same axis to be rotated about more than once.

No matter which convention is chosen, Euler angles specify three degrees of freedom with three parameters. However, they suffer a possible loss of degrees of freedom due to gimbal lock. Therefore, representations with more than 3 degrees of freedom are often chosen to parameterize rotations. However, these representations must also introduce constraints to ensure that there are only three degrees of freedom. For

further information see [84, 95].

One representation is the rotation matrix (also known as a direction cosine matrix). Rotations about the z, y and x axis are specified by:

$$\mathbf{R}_{\psi} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{C.1}$$

$$\mathbf{R}_{\theta} = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix} \tag{C.2}$$

$$\mathbf{R}_{\psi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix} \tag{C.3}$$

A general rotation matrix can be constructed by subsequent matrix multiplication of the above matrices. If a body fixed rotation is to be performed about axis a, b, c then the rotation matrix is:

$$\mathbf{R}_{abc} = \mathbf{R}_c\mathbf{R}_b\mathbf{R}_a \tag{C.4}$$

If a space fixed rotation is to be performed then the ordering is:

$$\mathbf{R}_{abc} = \mathbf{R}_a\mathbf{R}_b\mathbf{R}_c \tag{C.5}$$

This matrix is constrained by the equation:

$$\mathbf{R}^{-1} = \mathbf{R}^T \tag{C.6}$$

Which implies that $\mathbf{R}$ is orthonormal (i.e. $\mathbf{R}\mathbf{R}^T = \mathbf{I}$). Since it is orthonormal its determinant is found below.

$$
\begin{aligned}
(\det \mathbf{R})^2 &= \det \mathbf{R} \det \mathbf{R}^T && \text{(C.7)} \\
&= \det \mathbf{R}\mathbf{R}^T && \text{(C.8)} \\
&= \det I && \text{(C.9)} \\
&= = 1 && \text{(C.10)} \\
\Rightarrow \det \mathbf{R} &= \pm 1 && \text{(C.11)}
\end{aligned}
$$

A determinant of $+1$ corresponds to a rotation matrix, while a determinant of -1 corresponds to a rotation matrix. It is therefore good practice to check that a given rotation matrix does in fact have a $+1$ determinant.

The upper diagonal of $\mathbf{R}^{-1} = \mathbf{R}^T$ results in six constraint equations for the nine elements of the rotation matrix, which correctly allows for three degrees of freedom in rotation.

C.1.2 Quaternions

Quaternions are a four element parameterization of rotation that represent a rotation $\theta : 0 \leq \theta \leq 2\pi$ about a unit eigenvector of rotation $\mathbf{e} = [e_1, e_2, e_3]^T$ in an inertial frame that is defined by unit vectors $[\mathbf{i}, \mathbf{j}, \mathbf{k}]$ [84, 96]. Note that due to Hamilton:

$$
\begin{aligned}
\mathbf{i}^2 &= \mathbf{j}^2 = \mathbf{k}^2 = 1 && \text{(C.12)} \\
\mathbf{ij} &= -\mathbf{ji} = \mathbf{k} && \text{(C.13)} \\
\mathbf{jk} &= -\mathbf{kj} = \mathbf{i} && \text{(C.14)} \\
\mathbf{ki} &= -\mathbf{ik} = \mathbf{j} && \text{(C.15)}
\end{aligned}
$$

The quaternion is represented as:

$$\mathbf{q} = \begin{bmatrix} \bar{\mathbf{q}} \\ q_4 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} e_1 \sin\frac{\theta}{2} \\ e_2 \sin\frac{\theta}{2} \\ e_3 \sin\frac{\theta}{2} \\ \cos\frac{\theta}{2} \end{bmatrix} \tag{C.16}$$

(C.17)

The conjugate rotation is $\mathbf{q}^* = [-\bar{\mathbf{q}}, q_4]^T$.

A single constraint is necessary for the quaternion representation to have only three degrees of freedom. The constraint that the 2-norm of the quaternion has a value of 1 serves this purpose and is listed below.

$$|\mathbf{q}| = q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1 \tag{C.18}$$

Note that by convention, $q_4 > 0$.

In comparison to a rotation $\mathbf{R}_a = \mathbf{R}_b\mathbf{R}_c$, a sequence of rotations using a quaternion may be written as:

$$\begin{bmatrix} q_1^a \\ q_2^a \\ q_3^a \\ q_4^a \end{bmatrix} = \begin{bmatrix} q_4^b & q_3^b & -q_2^b & q_1^b \\ -q_3^b & q_4^b & -q_1^b & q_2^b \\ q_2^b & -q_1^b & -q_4^b & q_3^b \\ -q_1^b & -q_2^b & -q_3^b & q_4^b \end{bmatrix} \begin{bmatrix} q_1^c \\ q_2^c \\ q_3^c \\ q_4^c \end{bmatrix} \tag{C.19}$$

$$\mathbf{q}_a = \mathbf{q}_b \otimes \mathbf{q}_c \tag{C.20}$$

The transformation between from a quaternion to a transformation matrix is given by:

$$\mathbf{R}(\mathbf{q}) = (q4^2 - \bar{\mathbf{q}}^T\bar{\mathbf{q}})\mathbf{I} + 2\bar{\mathbf{q}}\bar{\mathbf{q}}^T - 2q_4\mathbf{Q} \tag{C.21}$$

$$\mathbf{Q} = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \tag{C.22}$$

$$\tag{C.23}$$

A Matlab program for converting from a rotation matrix to a quaterion is given at the end of Appendix B.1.

C.1.3 Modified Rodrigues Parameters

Modified rodrigues parameters $\mathbf{p}$ [65] is a 3 element vector for rotation. It can be determined from a quaternion parameterization ($\mathbf{e}$ is again the eigenvector of rotation and the rotation about this vector is $\theta : 0 \leq \theta \leq 2\pi$).

$$\mathbf{p} = \frac{1}{1+q_4}\bar{\mathbf{q}} \tag{C.24}$$

$$= \mathbf{e}\tan\left(\frac{\theta}{4}\right) \tag{C.25}$$

$$= \frac{1}{4}\mathbf{a}_p \tag{C.26}$$

In order to convert this back to a quaternion:

$$\mathbf{q}(\mathbf{a}_p) = \frac{1}{16 + \mathbf{a}_p^T\mathbf{a}_p}\begin{bmatrix} 8\mathbf{a}_p \\ 16 - \mathbf{a}_p^T\mathbf{a}_p \end{bmatrix} \tag{C.27}$$

The vector $\mathbf{a}_p$ is used as the representation for the error quaternion for the Multiplicative Extended Kalman Filter in Section 3.6.

C.2 Deterministic Relationships between Gaussian Distributions

In the previous section, different representations of rotation were introduced that contain deterministic or redundant variables, in order to avoid singularities (i.e. gimbal lock). In this section, the effect of deterministic variables in a multivariate Gaussian distribution is discussed. It is proved here that one of the eigenvalues will become zero. This is problematic for numerical estimators, since slight numerical errors can cause this eigenvalue to become slightly less than zero, creating a covariance matrix that is negative definite, which no longer has any physical meaning since it can correspond to probabilities that are greater that one.

Multivariate Gaussian distributions are commonly used in a number of recursive estimators such as the Extended Kalman Filter. These distributions are parameterized as a mean μ and a covariance matrix Λ, which is symmetric positive definite by definition. The variance of each variable $\mathbf{X_i}$ is represented as σ_i^2, while the covariance between to variables $\mathbf{X}_i$ and $\mathbf{X}_j$ is $\rho\sigma_i\sigma_j$, where ρ is the correlation coefficient.

$$\begin{aligned} \mathbf{X} &= [X_1 \dots X_n]^T && \text{(C.28)} \\ E[\mathbf{X}] &= \mu && \text{(C.29)} \\ E[\mathbf{X}^T\mathbf{X}] &= \Lambda && \text{(C.30)} \\ E[X_i^2] &= \sigma_i^2 && \text{(C.31)} \\ E[X_iX_j] &= \rho\sigma_i\sigma_j && \text{(C.32)} \end{aligned}$$

The probability distribution of n Gaussian random variables is represented as:

$$\begin{aligned} p(\mathbf{X}=\mathbf{x}) &= \mathbf{N}(\mu, \mathbf{\Lambda}) && \text{(C.33)} \\ &= \frac{1}{(2\pi)^{n/2}|\Lambda|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T\Lambda^{-1}(\mathbf{x}-\mu)} && \text{(C.34)} \end{aligned}$$

From this representation, it is evident that the contours of equal probability form an ellipsoid in n-dimensional space. The axes of the ellipsoid are the eigenvectors of Λ and the relative widths of the ellipsoid are the eigenvalues of Λ.

In the case where one of the variables in the distribution is not independent of all of the other variables in the distribution (i.e. one variable is entirely dependent on other variables), it is no longer a random variable and becomes deterministic. This results in the loss of rank of the covariance matrix (i.e. making it symmetric positive semi-definite). This implies that one of the eigenvalues of the covariance matrix becomes zero.

The two variable example can show the connection between the rank, eigenvalue and conditional dependence. To express this in probabilistic terms the variable x_1 is a random variable and x_2 is a deterministic functions of x_1. The correlation coefficient is ρ.

$$\mathbf{\Lambda} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix} \tag{C.35}$$

$$\tag{C.36}$$

To compute the eigenvalues of the covariance matrix:

$$\det(\lambda - \mathbf{\Lambda}) = 0 = \det \begin{bmatrix} \lambda - \sigma_1^2 & -\rho\sigma_1\sigma_2 \\ -\rho\sigma_1\sigma_2 & \lambda - \sigma_2^2 \end{bmatrix} \tag{C.37}$$

$$0 = \det(\lambda - \sigma_1^2)\det(\lambda - \sigma_2^2 - \rho^2\sigma_1^2\sigma_2^2(\lambda - \sigma_1^2)^{-1}) \tag{C.38}$$

$$0 = \lambda^2 - (\sigma_1^2 + \sigma_2^2) - (\rho^2 - 1)\sigma_1^2\sigma_2^2 \tag{C.39}$$

$$\lambda = \frac{1}{2}(\sigma_1^2 + \sigma_2^2) \pm \frac{1}{2}\sqrt{(\sigma_1^2 + \sigma_2^2)^2 + 4(\rho^2 - 1)\sigma_1^2\sigma_2^2} \tag{C.40}$$

Since x_2 is a deterministic functions of x_1, $\rho = \pm 1$. Now

$$\begin{aligned}
\lambda &= \frac{1}{2}(\sigma_1^2 + \sigma_2^2) \pm \frac{1}{2}\sqrt{(\sigma_1^2 + \sigma_2^2)^2 + 4(\rho^2 - 1)\sigma_1^2\sigma_2^2} && \text{(C.41)} \\
&= \frac{1}{2}(\sigma_1^2 + \sigma_2^2) \pm \frac{1}{2}\sqrt{(\sigma_1^2 + \sigma_2^2)^2} && \text{(C.42)} \\
&= \{(\sigma_1^2 + \sigma_2^2), 0\} && \text{(C.43)}
\end{aligned}$$

This shows that a deterministic variable leads to an eigenvalue of 0, which corresponds to a loss of full rank of the covariance matrix. This can also be seen from the perspective of conditional probability. Using the formulas for computing the conditional probability of a multivariable gaussian distribution:

$$\begin{aligned}
p(x_2|x_1) &= \mathbf{N}\left(\mu_2 + \frac{\rho\sigma_1\sigma_2}{\sigma_1^2}(x_1 - \mu_1), \sigma_2^2 - \frac{\rho^2\sigma_1^2\sigma_2^2}{\sigma_1^2}\right) && \text{(C.44)} \\
&= \mathbf{N}\left(\mu_2 + \frac{\rho\sigma_2}{\sigma_1}(x_1 - \mu_1), \sigma_2^2 - \rho^2\sigma_2^2\right) && \text{(C.45)} \\
& && \text{(C.46)}
\end{aligned}$$

Again, using the fact that $\rho = 1$:

$$\begin{aligned}
p(x_2|x_1) &= \mathbf{N}\left(\mu_2 + \frac{\sigma_2}{\sigma_1}(x_1 - \mu_1), 0\right) && \text{(C.47)} \\
& && \text{(C.48)}
\end{aligned}$$

This implies that there is no uncertainty in the value of x_2 given x_1.

C.3 Estimation of Velocity from a Sequence of Position Measurements

The multiplicative Extended Kalman Filter that is developed in Section 3.6 estimates the linear and angular velocity based only on linear and angular position measurements. In this section, the details of this type of estimation are investigated for the scalar case of estimating linear velocity from position measurements (similar arguments apply to angular velocity and position). The dynamic model to be estimated in this example is:

$$\begin{bmatrix} \dot{r} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} r \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} n_w \tag{C.49}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} r \\ v \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} n_v \tag{C.50}$$

Where the noise, n_w and n_v is modeled as independent markov white gaussian noise with variances σ_w^2 and σ_v^2 respectively.

Now, considering the continuous time minimum mean squared error estimator (Continuous Time Kalman Filter, see [29]), the state estimates are:

$$\begin{bmatrix} \dot{\hat{r}} \\ \dot{\hat{v}} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{r} \\ \hat{v} \end{bmatrix} + \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} (y - \hat{y}) \tag{C.51}$$

$$\begin{bmatrix} \dot{\hat{r}} \\ \dot{\hat{v}} \end{bmatrix} = \begin{bmatrix} -L_1 & 1 \\ -L_2 & 0 \end{bmatrix} \begin{bmatrix} \hat{r} \\ \hat{v} \end{bmatrix} + \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} y \tag{C.52}$$

Where $[L_1, L_2]^T$ are the Kalman Gain which are by definition the solution to the following filter form of the matrix differential Riccati equation for the error covariance $\mathbf{Q}$:

$$\dot{\mathbf{Q}} = \begin{bmatrix} \dot{q}_1 & \dot{q}_2 \\ \dot{q}_2 & \dot{q}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} q_1 & q_2 \\ q_2 & q_3 \end{bmatrix} + \begin{bmatrix} q_1 & q_2 \\ q_2 & q_3 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}^T + \begin{bmatrix} 0 & 0 \\ 0 & \sigma_w^2 \end{bmatrix} \quad \text{(C.53)}$$

$$+ \begin{bmatrix} q_1 & q_2 \\ q_2 & q_3 \end{bmatrix} \begin{bmatrix} \sigma_v^{-2} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} q_1 & q_2 \\ q_2 & q_3 \end{bmatrix} \quad \text{(C.54)}$$

$$\begin{bmatrix} L_1 \\ L_2 \end{bmatrix} = \begin{bmatrix} q_1 & q_2 \\ q_2 & q_3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \sigma_v^{-2} \quad \text{(C.55)}$$

For the steady state case, the solutions for the error covariances are:

$$q_1 = \sqrt{2}\sigma_v^2 \sqrt{\frac{\sigma_w}{\sigma_v}} \quad \text{(C.56)}$$

$$q_2 = \sigma_v \sigma_w \quad \text{(C.57)}$$

$$q_3 = \sqrt{2}\sqrt{\frac{\sigma_w}{\sigma_v}} \sigma_v \sigma_w \quad \text{(C.58)}$$

The Kalman gains can now be found as:

$$L_1 = \sqrt{2}\sqrt{\frac{\sigma_w}{\sigma_v}} \quad \text{(C.59)}$$

$$L_2 = \frac{\sigma_w}{\sigma_v} \quad \text{(C.60)}$$

These clearly only depend on the ratio between the process and measurement noise.

The transfer function from the position measurements to the velocity estimates can be found by converting Equation C.49 to transfer function form:

$$\frac{V(s)}{Y(s)} = \frac{L_2 s}{s^2 + L_1 s + L_2} \quad \text{(C.61)}$$

This is a second order differential equation with a natural frequency of $\sqrt{\frac{\sigma_w}{\sigma_v}}$, a

damping ratio of $\frac{1}{\sqrt{2}}$ and a zero at $s = 0$. This result is interesting as it implies that the estimator that minimizes the trace of the state estimate covariance matrix of the state is based only on the first and second derivatives in the continuous time case (or only two previous samples in the discrete time case).

The bode plot of this transfer function is compared in Figure C-1 to that of a pure differentiator $G(s) = s$, which is physically unrealizable. A ratio of $\sigma_w/\sigma_v = 900$ was chosen as it is close to the ratio used for position. This shows very close tracking to a continuous time differentiator up until the natural frequency.

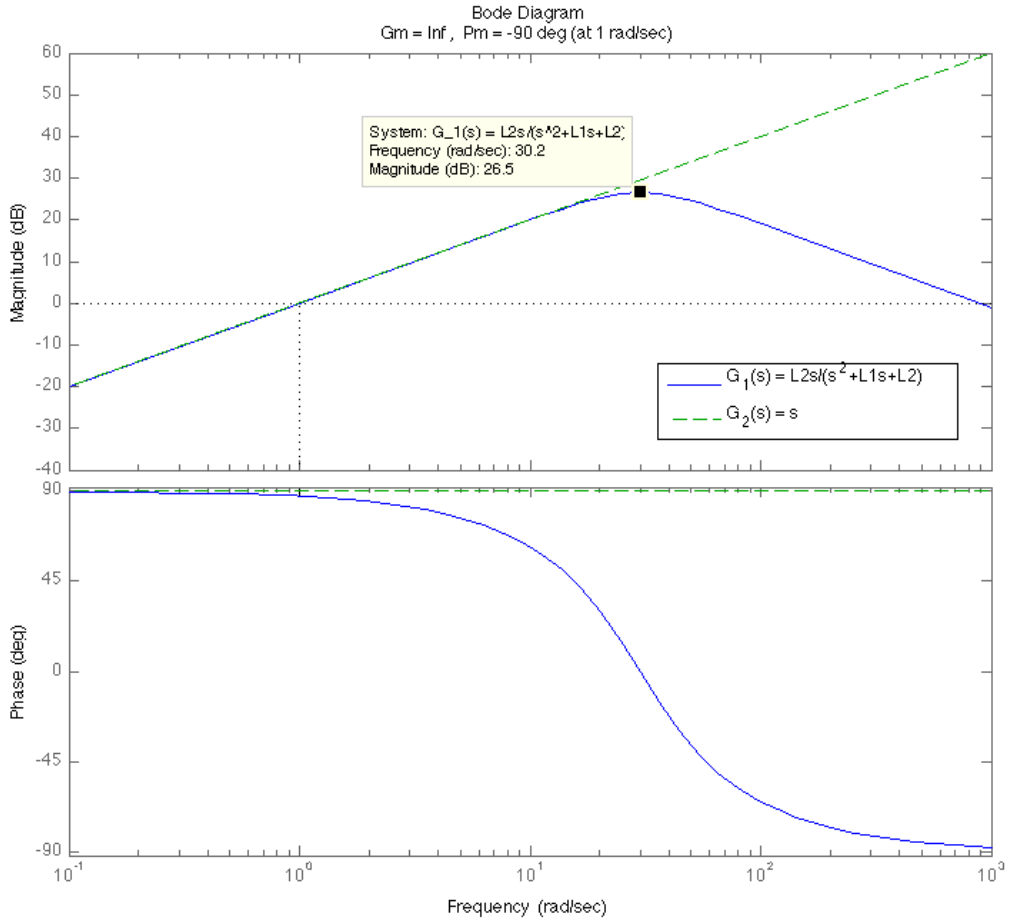


Figure C-1: Bode Plot Comparison of Steady State Continous Time Kalman Filter to Differentiator

Additionally, there is the issue of aliasing in the implementation of a discrete time filter for Equation C.61. As a result, any high frequency noise in the input will be aliased to a lower frequency. Figure C-1 shows that low frequency signals

are amplified, which means that high frequency noise in position measurements can be incorrectly interpreted as low frequency velocities due to aliasing. This effect is observed in Section 3.7.3.

C.4 Levenberg-Marquardt Method for Nonlinear Least Squares

The Levenberg-Marquardt Algorithm is a method for iteratively solving the nonlinear least squares problem. The presentation in this section is based on Van Den Bos's description [91]. In this problem let θ be a set of parameters that are to be estimated, and ω be a set of n measurements of these parameters that are related by an arbitrary nonlinear function $g(\theta)$.

$$\omega_n = E[g_n(\theta)] \tag{C.62}$$

The general non-linear least squares problem requires the minimization of a cost function $J(\theta)$.

$$J(\theta) = \sum_n \left(\omega_n - g_n(\theta)\right)^2 \tag{C.63}$$

In order to solve this problem a Jacobian matrix X and an error $d(\theta)$ is defined. Additionally, R is a symmetric positive definite weighting matrix that is nominally the inverse of the covariance matrix C of each of the measurements ω. Using this the weighted least squares cost is redefined for a candidate increment $\Delta\theta$ away from a previous iteration's solution θ_c.

$$X = \frac{\partial g(\theta)}{\partial \theta^T}\Big|_{\theta=\theta_c} \tag{C.64}$$

$$d(\theta_c) = \omega - g(\theta_c) \tag{C.65}$$

$$J(\theta_c + \Delta\theta) = [d(\theta_c) - X\Delta\theta]^T C^{-1} [d(\theta_c) - X\Delta\theta] \tag{C.66}$$

Since there is a possibility of $X^T C^{-1} X$ becoming singular, we include an equality constraint, where k is a positive constant.

$$||\Delta\theta||^2 = \sum_i (\Delta\theta_i)^2 = k^2 \tag{C.67}$$

As a result, the cost can be augmented with a Lagrange multiplier λ (also referred to as a damping factor), which can be minimized and the optimal θ_{LM} can be solved for.

$$\begin{aligned} \Delta\theta_{LM} &= \underset{\Delta\theta}{\operatorname{argmin}}\, J(\theta_c + \Delta\theta) + \lambda(||\Delta\theta||^2 - k^2) && \text{(C.68)} \\ 0 &= \frac{\partial J(\theta_c + \Delta\theta)}{\partial \Delta\theta} + 2\lambda||\Delta\theta|| && \text{(C.69)} \\ \Rightarrow \lambda &= -\frac{1}{2||\Delta\theta||}\frac{\partial J(\theta_c + \Delta\theta)}{\partial \Delta\theta} && \text{(C.70)} \end{aligned}$$

The main iterative step can now be defined below.

$$\Delta\theta_{LM} = (X^T R X + \lambda I)^{-1} X^T R d(\theta) \tag{C.71}$$

Pseudo-code for this algorithm is presented as follows.

Algorithm 3 Levenberg-Marquardt Algorithm

$\theta_c \leftarrow$ initial guess
$\lambda \leftarrow$ initial value
$\nu \leftarrow 1$
while termination **do**
  $\lambda_c \leftarrow \lambda/\nu$
  $X = \frac{\partial g(\theta)}{\partial \theta^T}\Big|_{\theta=\theta_c}$
  $\Delta\theta_{LM} = (X^T R X + \lambda I)^{-1} X^T R d(\theta_c)$
  $J(\theta_c + \Delta\theta) = [d(\theta_c) - X\Delta\theta]^T C^{-1} [d(\theta_c) - X\Delta\theta]$
  if $J(\theta_c + \Delta\theta) < J(\theta_c)$ **then**
    $\theta_c \leftarrow \theta_c + \Delta\theta_{LM}$
    $\lambda \leftarrow \lambda/\nu$
  else
    $\lambda \leftarrow \lambda$
    $\Delta\theta_{LM} = (X^T R X + \lambda I)^{-1} X^T R d(\theta_c)$
    $J(\theta_c + \Delta\theta) = [d(\theta_c) - X\Delta\theta]^T C^{-1} [d(\theta_c) - X\Delta\theta]$
    if $J(\theta_c + \Delta\theta) < J(\theta_c)$ **then**
      $\theta_c \leftarrow \theta_c + \Delta\theta_{LM}$
      $\lambda \leftarrow \lambda$
    else
      repeat
        $\lambda \leftarrow \lambda\nu$
        $\Delta\theta_{LM} = (X^T R X + \lambda I)^{-1} X^T R d(\theta_c)$
        $J(\theta_c + \Delta\theta) = [d(\theta_c) - X\Delta\theta]^T C^{-1} [d(\theta_c) - X\Delta\theta]$
      until $J(\theta_c + \Delta\theta) < J(\theta_c)$
      $\theta_c \leftarrow \theta_c + \Delta\theta_{LM}$
    end if
  end if
end while

Bibliography

[1] ISS016E014220. `http://ssl.mit.edu/spheres/pictures/ISS10/ISS016E014220.jpg`, 2008.

[2] cvBlobsLib. `http://opencv.willowgarage.com/wiki/cvBlobsLib`, November 2009.

[3] MIT SPHERES Satellites. `http://ssl.mit.edu/spheres`, 2009.

[4] F. Ababsa, M. Mallem, and D. Roussel. Comparison between particle filter approach and kalman filter-based technique for head tracking in augmented reality systems. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 1, pages 1021–1026 Vol.1, April-1 May 2004.

[5] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):698–700, 1987.

[6] Y. Bar-Shalom and Xiao-Rong Li. *Estimation and Tracking*. Artech House Inc., 1993.

[7] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern and Machine Intelligence*, 14(2), 1992.

[8] Jeffrey J. Biesiadecki, Chris Leger, and Mark W. Maimone. Tradeoffs between directed and autonomous driving on the mars exploration rovers. In Sebastian Thrun, Rodney A. Brooks, and Hugh F. Durrant-Whyte, editors, *ISRR*, volume 28 of *Springer Tracts in Advanced Robotics*, pages 254–267. Springer, 2005.

[9] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, Cambridge, MA, 2008.

[10] D.C. Brown. Decentering distortion of lenses. 32(3):444–462, 1966.

[11] Robert G. Brown and Patrick Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering*. Wiley, 3rd edition, 1996.

[12] Olivier Brun, Vincent Teuliere, and Jean-Marie Garcia. Parallel particle filtering. *Journal of Parallel and Distributed Computing*, 62(7):1186 – 1202, 2002.

[13] Ian Buck. GPU computing with NVIDIA CUDA. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 6, New York, NY, USA, 2007. ACM.

[14] Robert Burtch. History of photogrammetry. Technical report, Ferris State University, 2008.

[15] Stefano Carpin and Enrico Pagello. On parallel rrts for multi-robot systems. In *Proc. 8th Conf. Italian Association for Artificial Intelligence*, pages 834–841, 2002.

[16] Javier Civera, Andrew J. Davison, and J. M. M. Montiel. Inverse depth parametrization for monocular slam. *IEEE Transactions on Robotics*, 24(5):932–945, 2008.

[17] J. Crassidis, R. Alonso, and J. L. Junkins. Optimal Attitude and Position Determination from Line-of-Sight Measurements. In *Special Battin Issue of the Journal of the Astronautical Sciences*, 2000.

[18] John L. Crassidis and John L. Junkins. *Optimal Estimation of Dynamic Systems*. Chapman and Hall, 2004.

[19] John L. Crassidis and F. Landis Markley. Predictive filtering for nonlinear systems. *Journal of Guidance, Control, and Dynamics*, 20:566–572, 1997.

[20] DARPA and Boeing. ASTRO Captures NextSat. `http://www.boeing.com/ids/phantom_works/orbital/oe_057.html`, July 2007.

[21] Andrew J. Davison and Nobuyuki Kita. Sequential localisation and map-building for real-time computer vision and robotics. *Robotics and Autonomous Systems*, 36(4):171–183, 2001.

[22] Adam Deslauriers, Chad English, Chris Bennett, Peter Iles, Ross Taylor, and Andrew Montpool. 3d inspection for the shuttle return to flight. In Richard T. Howard and Robert D. Richards, editors, *Spaceborne Sensors III*, volume 6220, page 62200H. SPIE, 2006.

[23] Nathaniel Fairfield. *Localization, Mapping, and Planning in 3D Environments*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 2009.

[24] Wigbert Fehse. *Automated Rendezvous and Docking of Spacecraft*. Cambridge University Press, 2003.

[25] Paul D. Fiore. Efficient linear solution of exterior orientation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(2):140–148, 2001.

[26] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[27] Steven E. Fredrickson, Larry W. Abbott, Steve Duran, J. David Jochim, J. William Studak, Jennifer D. Wagenknecht, and Nichole M. Williams. Mini AERCam: Development of a Free-Flying Nanosatellite Inspection Robot. In Peter Tchoryk, Jr., and James Shoemaker, editors, *Space Systems Technology and Operations*, volume 5088, pages 97–111. SPIE, 2003.

[28] Lance B. Gatrell, William A. Hoff, and Cheryl W. Sklair. Robust image features: concentric contrasting circles and their image extraction. volume 1612, pages 235–244. SPIE, 1992.

[29] Arthur Gelb, editor. *Applied Optimal Estimation*. The MIT Press, 1974.

[30] Joel Gibson and Oge Marques. Stereo depth with a unified architecture gpu. *Computer Vision and Pattern Recognition Workshop*, 0:1–6, 2008.

[31] Jerry Ginsberg. *Advanced Engineering Dynamics*. Cambridge University Press, 1998.

[32] S.B. Goldberg, M.W. Maimone, and L. Matthies. Stereo vision and rover navigation software for planetary exploration. In *Aerospace Conference Proceedings, 2002. IEEE*, volume 5, pages 5–2025–5–2036 vol.5, 2002.

[33] Daniel P. Goodwin, Laura E. Hembree, Joseph P. Curran, David S. Moyer, Russell L. Strachan, Ian Mills, and Jean-Sebastian Valois. Orbiter space vision system on space shuttle flight sts-80. volume 3074, pages 18–28. SPIE, 1997.

[34] Armin Gruen and Thomas S. Huang. *Calibration and Orientation of Cameras in Computer Vision*. Springer, 2001.

[35] K. Gunman, D. Hughes, J. L. Junkins, and N. Kehtarnaraz. A Vision Based DSP Embedded Navgation Sensor. *IEEE Journal of Sensors*, 2(5):428–442, 2002.

[36] Ziyad S. Hakura and Anoop Gupta. The design and analysis of a cache architecture for texture mapping. *SIGARCH Comput. Archit. News*, 25(2):108–120, 1997.

[37] R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision*. Addison Wesley Longman, 2002.

[38] R.M. Haralick, C.N. Lee, X. Zhuang, V.G. Vaidya, and M.B. Kim. Pose Estimation from Corresponding Point Data. pages 258–263, 1987.

[39] Robert M. Haralick, Chung-Nan Lee, Karsten Ottenberg, and Michael Nölle. Review and analysis of solutions of the three point perspective pose estimation problem. *Int. J. Comput. Vision*, 13(3):331–356, 1994.

[40] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[41] Gustaf Hendeby, Jeroen Hol, Rickard Karlsson, and Fredrik Gustafsson. Graphics processing unit implementation of the particle filter. Technical Report LiTH-ISY-R-2749, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden, October 2006.

[42] C. Glen Henshaw, L. Henshaw, and S. Roderick. LIIVe: A Small, Low-Cost Autonomous Inspection Vehicle. In *AIAA SPACE 2009 Conference and Exposition*, AIAA 2009-6544, 2009.

[43] B. K. P. Horn. Tsai's Camera Calibration Method Revisited. Technical report, Massachusetts Institute of Technology, 2000.

[44] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America. A*, 4(4):629–642, Apr 1987.

[45] Berthold Klaus Paul Horn. *Robot Vision*. The MIT Press, 1986.

[46] Andrew Howard. Real-time stereo visual odometry for autonomous ground vehicles. In *Intelligent Robots and Systems, 2008. (IROS 2008)*.

[47] R.T. Howard, A.F. Heaton, R.M. Pinson, and C.K. Carrington. Orbital express advanced video guidance sensor. In *Aerospace Conference, 2008 IEEE*, pages 1–10, March 2008.

[48] T.S. Huang and A.N. Netravali. Motion and structure from feature correspondences: a review. *Proceedings of the IEEE*, 82(2):252–268, Feb 1994.

[49] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *IWAR '99: Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, page 85, Washington, DC, USA, 1999. IEEE Computer Society.

[50] J. Kawaguchi, A. Fujiwara, and T. Uesugi. Hayabusa (MUSES-C) - Rendezvous and Proximity Operation. In *56th International Astronautical Congress*, IAC-05-A3.5.A.01, 2005.

[51] Son-Goo Kim, John L. Crassidis, yang Cheng, Adam M. Fosbury, and John L. Junkins. Kalman filtering for relative spacecraft attitude and position estimation. *AIAA Journal of Guidance, Control and Dynamics*, 30(1):133–143, Jan-Feb 2007.

[52] Soon-Go Kim, John Crassidis, Yang Cheng, and John Junkins. Kalman Filtering for Relative Spacecraft Attitude and Position Estimation. In *AIAA Guidance, Navigation and Control Conference and Exhibition*, AIAA-2005-6087, 2005.

[53] B. Kisacanin, S. S. Bhattacharyya, and S. Chai. *Embedded Computer Vision*. Springer, 2009.

[54] T. Kubota, T. Hashimoto, J. Kawaguchi, M. Uo, and K. Shirakawa. Guidance and navigation of hayabusa spacecraft for asteroid exploration and sample return mission. In *SICE-ICASE, 2006. International Joint Conference*, pages 2793–2796, Oct. 2006.

[55] C. Langis, M. Greenspan, and G. Godin. The parallel iterative closest point algorithm. *3D Digital Imaging and Modeling, International Conference on*, 0:195, 2001.

[56] Louis J. Lanzerotti. Assessment of options for extending the life of the hubble space telescope: Final report. Technical report, National Research Council of the National Academies, 2005.

[57] A.G. Ledebuhr, L.C. Ng, M.S. Jones, B.A. Wilson, R.J. Gaughan, E.F. Breitfeller, W.G. Taylor, J.A. Robinson, D.R. Antelman, and D.P. Nielsen. Microsatellite ground test vehicle for proximity and docking operations development. In *Aerospace Conference, 2001, IEEE Proceedings.*, volume 5, pages 2493–2504 vol.5, 2001.

[58] Pascal Lee, NASA, and JAXA. Size of Itokawa and ISS, 2006.

[59] E.J. Lefferts, F.L. Markley, and M.D. Shuster. Kalman Filtering for Spacecraft Attitude Estimation. *AIAA 20th Aerospace Sciences Meeting*, 1982.

[60] John Leonard and Hugh Durrant-Whyte. *Directed Sonar Sensing for Mobile Robot Navigation*. Kluwer Academic Publishers, 1992.

[61] Yi Ma, Stefano Soatto, Jana Kosecka, and S. Shankar Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer, 2006.

[62] S. G. MacLean and H. F. L. Pinkney. Machine Vision in Space. *Canadian Aeronautics and Space Journal*, 29(2):63–77, 1993.

[63] Mark Maimone, Yang Cheng, and Larry Matthies. Two years of visual odometry on the mars exploration rovers: Field reports. *J. Field Robot.*, 24(3):169–186, 2007.

[64] Mark W. Maimone, Andrew Edie Johnson, Yang Cheng, Reg G. Willson, and Larry Matthies. Autonomous Navigation Results from the Mars Exploration Rover (MER) Mission. In Marcelo H. Ang Jr. and Oussama Khatib, editors, *ISER*, Springer Tracts in Advanced Robotics, pages 3–13. Springer, 2004.

[65] F. Landis Markley. Attitude Error Representations for Kalman Filtering. *Journal of Guidance, Control, and Dynamics*, 2003.

[66] Oscar Mateo Lozano and Kazuhiro Otsuka. Real-time visual tracker by stream processing. *Journal of Signal Processing Systems*, 57(2):285–295, November 2009.

[67] Matlab. Real-time workshop. `http://www.mathworks.com/products/rtw/`, 2009.

[68] Larry Matthies. *Dynamic Stereo Vision*. PhD thesis, Carnegie Mellon University, 1989.

[69] Larry Matthies, Mark Maimone, Andrew Johnson, Yang Cheng, Reg Willson, Carlos Villalpando, Steve Goldberg, Andres Huertas, Andrew Stein, and Anelia Angelova. Computer Vision on Mars. *International Journal of Computer Vision*, 2007.

[70] Catharine L. R. McGhan, Rebecca L. Besser, Robert M. Sanner, and Ella M. Atkins. Semi-Autonomous Inspection with a Neutral Buoyancy Free-Flyer. In *AIAA Guidance, Navigation and Control Conference and Exhibition*, AIAA 2006-6800, 2006.

[71] E. M. Mikhail, J. S. Bethel, and J.C. McGlone. *Introduction to Modern Photogrammetry*. John Wiley and Sins, Inc., 2001.

[72] Swait Mohan, Alvar Saenz-Otero, Simon Nolet, David W. Miller, and Steven Sell. Spheres flight operations testing and execution. *Acta Astronautica*, 2009.

[73] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002.

[74] NASA. STS-114 Shuttle Mission Imagery. `http://spaceflight.nasa.gov/gallery/images/shuttle/sts-114/html/s114e6521.html`, August 2005.

[75] Simon Nolet. The spheres navigation system: from early development to on-orbit testing. In *AIAA Guidance, Navigation and Control Conference and Exhibition*, AIAA-2007-6354, 2007.

[76] Jerome Obermark, Glenn Creamer, Bernard E. Kelm, William Wagner, and C. Glen Henshaw. SUMO/FREND: Vision System for Autonomous Satellite Grapple. In Richard T. Howard and Robert D. Richards, editors, *Sensors and Systems for Space Applications*, volume 6555, page 65550Y. SPIE, 2007.

[77] Clark F. Olson, Larry H. Matthies, Marcel Schoppers, and Mark W. Maimone. Rover navigation using stereo ego-motion. *Robotics and Autonomous Systems*, 43(4):215 – 229, 2003.

[78] L.M. Paz, P. Pinies, J.D. Tardos, and J. Neira. Large-scale 6-dof slam with stereo-in-hand. *IEEE Transactions on Robotics*, 24(5):946–957, Oct. 2008.

[79] Ioannis Pitas. *Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks*. Wiley, 1993.

[80] M. Romano. On-the-Ground Experiments of Autonomous Spacecraft Proximity-Navigation using Computer Vision and Jet Actuators. In *Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on*, pages 1011–1016, July 2005.

[81] M. Romano, D. A. Friedman, and T. J. Shay. Laboratory Experimentation of Autonomous Spacecraft Approach and Docking to a Collaborative Target. *Journal of Spacecraft and Rockets*, 44:164–173, January 2007.

[82] Fernando Sansò. An Exact Solution of the Roto-Translation Problem. *Photogrammetria*, 29(6):203–216, 1973.

[83] Taku Senoo, Yuji Yamakawa, Satoru Mizusawa, Akio Namiki, Masatoshi Ishikawa, and Makoto Shimojo. Skillful manipulation based on high-speed sensory-motor fusion. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 1611–1612, May 2009.

[84] Marcel J. Sidi. *Spacecraft Dynamics and Control: A Practical Engineering Approach*. Cambridge University Press, 1997.

[85] Cheryl W. Sklair, Lance B. Gatrell, William A. Hoff, and Michael Magee. Optical target location using machine vision in space robotics tasks. volume 1387, pages 380–391. SPIE, 1991.

[86] R. Smith, M. Self, and P. Cheeseman. Estimating Uncertain Spatial Relationships in Robotics. *Autonomous Robot Vehicles*, pages 167–193, 1990.

[87] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[88] Nikolas Trawny, Anastasios I. Mourikis, Stergios I. Roumeliotis, Andrew E. Johnson, and James F. Montgomery. Vision-aided inertial navigation for pin-point landing using observations of mapped landmarks: Research articles. *J. Field Robot.*, 24(5):357–378, 2007.

[89] R. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *Robotics and Automation, IEEE Journal of*, 3(4):323–344, August 1987.

[90] B. E. Tweddle. Real-time visual simultaneousl localization and mapping on a gpu. `http://beowulf.csail.mit.edu/18.337/projects/reports/tweddle_final_report.pdf`, 2009.

[91] Adriaan van den Bos. *Parameter Estimation for Engineers and Scientists*. John Wiley and Sons Inc., 2007.

[92] W. van der Mark and D.M. Gavrila. Real-time dense stereo for intelligent vehicles. *Intelligent Transportation Systems, IEEE Transactions on*, 7(1):38–50, March 2006.

[93] C. Villalpando and R. Some. Parallelizing Lunar Safe Landing Algorithms on the Tilera Tile 64 Processor. In *IEEE International Conference on Space Mission Challenges for Information Technology*, 2009.

[94] Carlos Villalpando. Acceleration of Stereo Correlation in Verilog. In *9th Annual Military and Aerospace Programmable Logic Device International Conference*, 2006.

[95] James R. Wertz, editor. *Spacecraft Attitude Determination and Control*. D. Reidel Publishing Company, 1985.

[96] Bong Wie. *Space Vehicle Dynamics and Control*. AIAA Education Series, 1998.

[97] Trevor Williams and Sergei Tanygin. On-Orbit Engineering Tests of the AER-Cam Sprint Robotic Camera Vehicle. In *Proceedings of the AAS/AIAA Space Flight Mechanics Meeting*, pages 1001–1020, 1998.

[98] David C. Woffinden and David K. Geller. Relative Angles-Only Navigation and Pose Estimation For Autonomous Orbital Rendezvous. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, AIAA 2006-6300, August 2008.

[99] W. Wu, L. Jin, J. Yang, P. Liu, and S. X. D. Tan. A Systematic Method For Functional Unit Power Estimation in Microprocessors. In *Design Automation Conference*, 2006.

[100] Hajime Yano, T. Kubota, H. Miyamoto, T. Okada, D. Scheeres, Y. Takagi, K. Yoshida, M. Abe, S. Abe, O. Barnouin-Jha, A. Fujiwara, S. Hasegawa, T. Hashimoto, M. Ishiguro, M. Kato, J. Kawaguchi, T. Mukai, J. Saito, S. Sasaki, and M. Yoshikawa. Touchdown of the Hayabusa Spacecraft at the Muses Sea on Itokawa. *Science*, 312(5778):1350–1353, 2006.

[101] Tetsuo Yoshimitsu, Jun'ichiro Kawaguchi, Tatsuaki Hashimoto, Takashi Kubota, Masashi Uo, Hideo Morita, and Kenichi Shirakawa. Hayabusa-final autonomous descent and landing based on target marker tracking. *Acta Astronautica*, 65(5-6):657 – 665, 2009.

[102] Zhengyou Zhang and Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1330–1334, 1998.

[103] Douglas Zimpfer, Peter Kachmar, and Seamus Tuohy. Autonomous Rendezvous, Capture and In-Space Assembly: Past, Present and Future. In *1st Space Exploration Conference: Continuing the Voyage of Discovery*, AIAA 2005-2523, 2005.